Exhibit 5

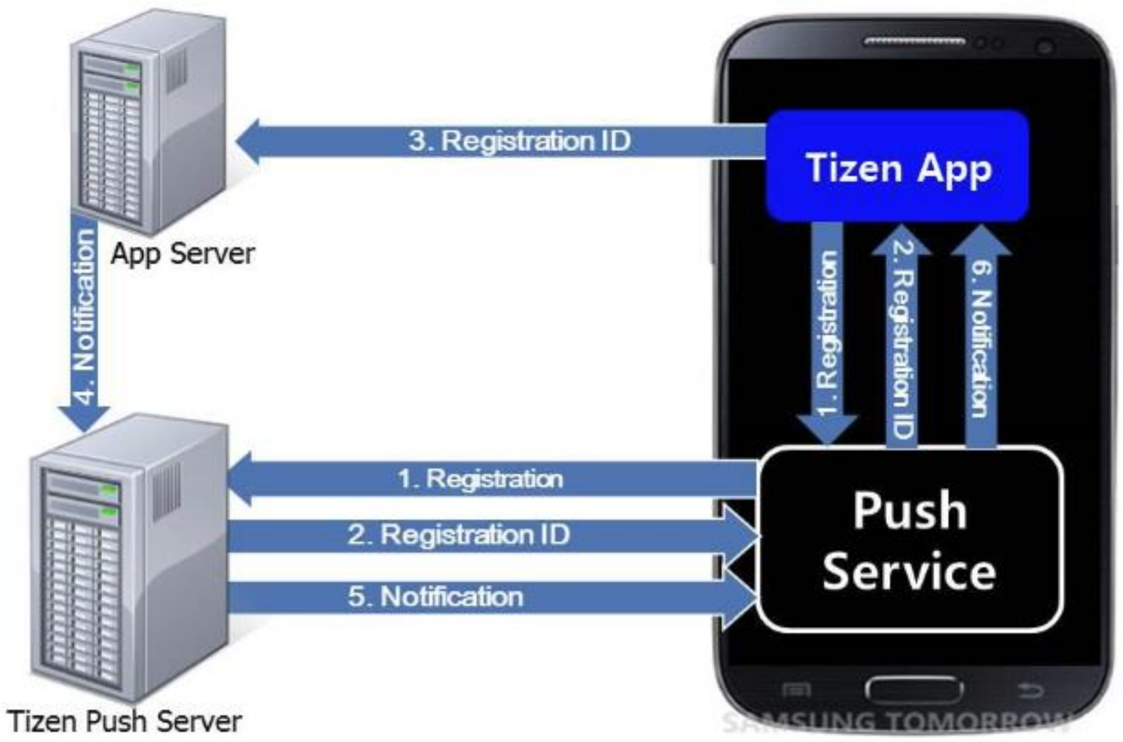
U.S. Patent No. 9,198,117 ("'117 Patent")

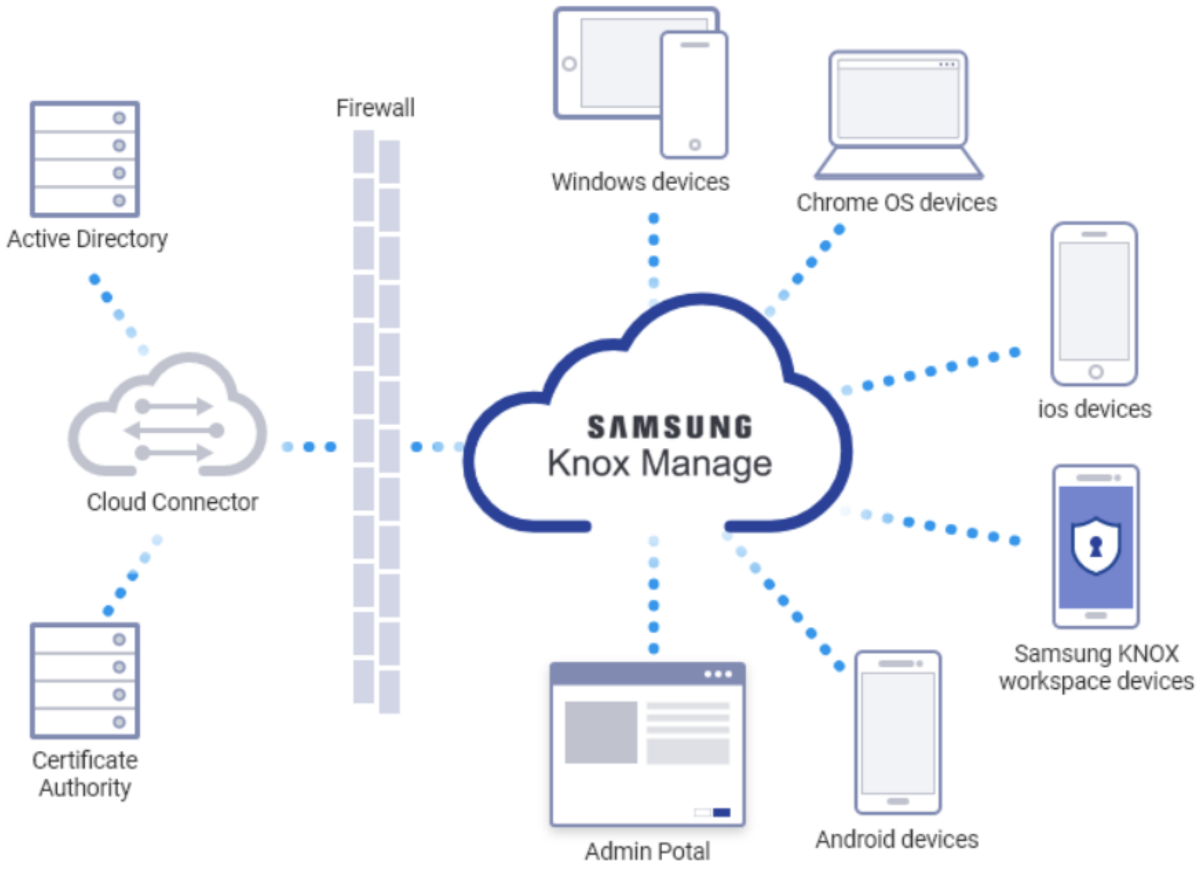
Accused Devices: Samsung's mobile electronic devices (e.g., Galaxy phones and tablets, as well as Samsung devices which include Samsung Knox functionality), and Samsung Tizen devices (e.g., TVs and wearables), and all versions and variations thereof since the issuance of the asserted patent.

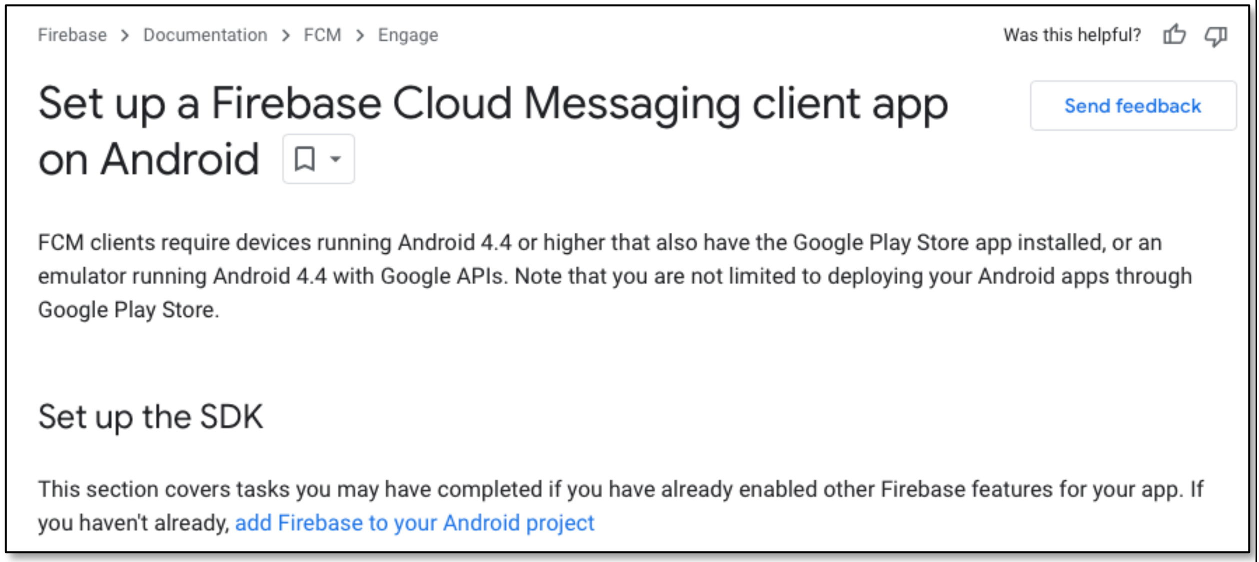
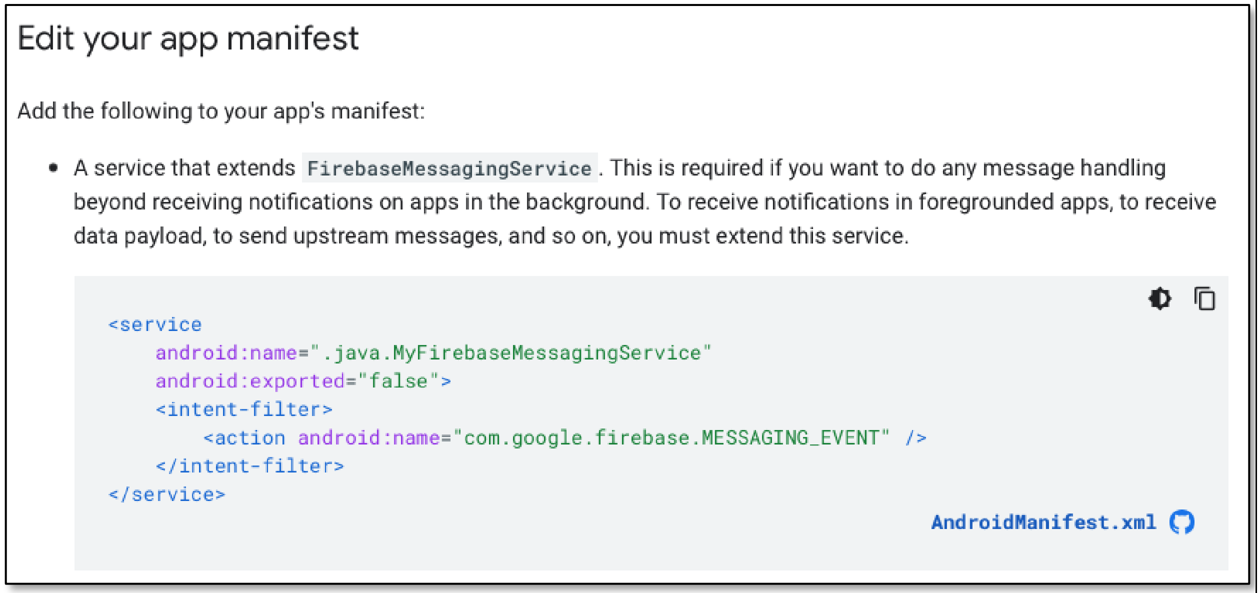
Claim 1

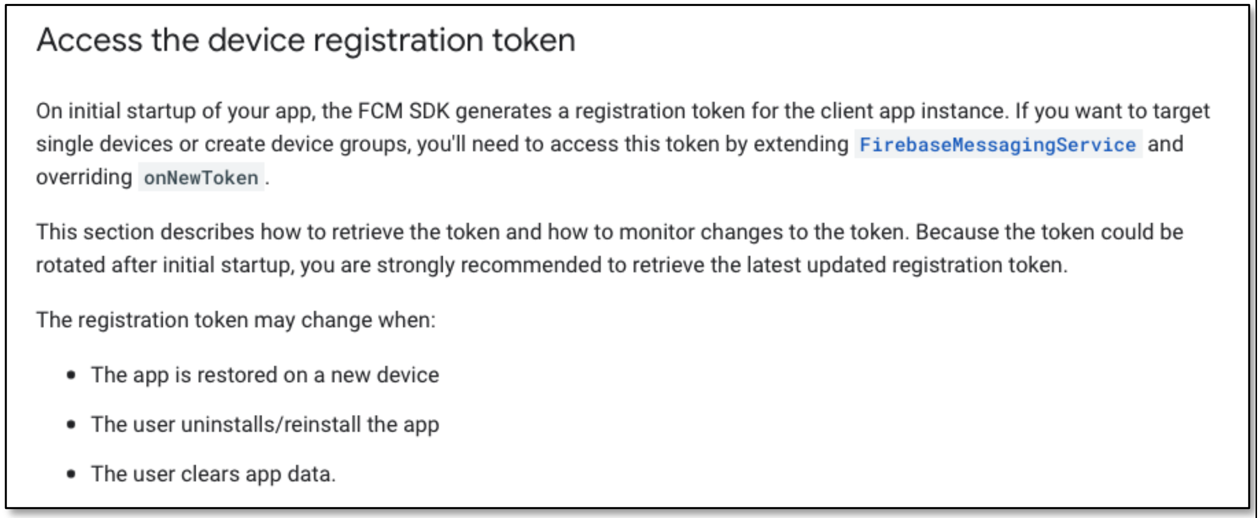
Issued Claim(s)	Public Documentation
[1pre]. A network system comprising:	Samsung's devices and push messaging servers comprise a "network system." *See, e.g.,* https://www.samsung.com/us/smartphones/galaxy-s22/;

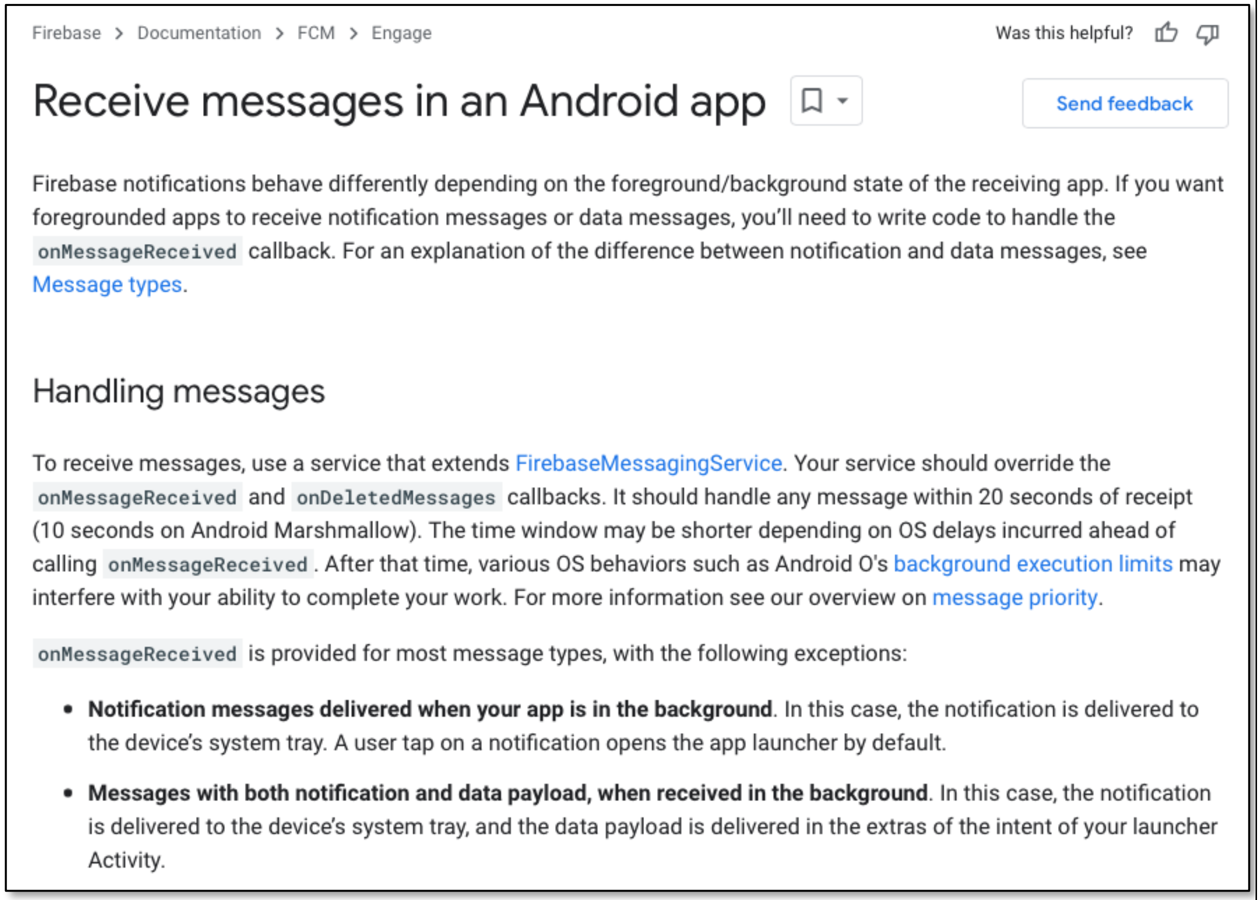
	 https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/;

	Architecture The architecture of the Tizen Push service is described in detail in the mobile native Push guide. **Figure: Service architecture**  To receive push notifications for your application: 1. Request permission to access the Tizen push servers for using the push service API. 2. Wait for a confirmation email for the request. 3. Register the installed application on the device. 4. Connect to the push service for receiving push notifications. 5. Receive notifications from the push service.

	https://docs.tizen.org/application/web/guides/messaging/push/. As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a "network system."  Firewall Active Directory Windows devices Chrome OS devices ios devices Cloud Connector SAMSUNG Knox Manage Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm
[1a] a plurality of device messaging agents, each executable on a respective one of a plurality of mobile end-user devices configured to exchange Internet	Samsung devices and push messaging servers comprise "a plurality of device messaging agents, each executable on a respective one of a plurality of mobile end-user devices configured to exchange Internet data via a data connection to a wireless network."

data via a data connection to a wireless network; and	For example, Samsung Galaxy phones and tablets comprise multiple device agents which have an identifier. *See e.g.*,  Firebase > Documentation > FCM > Engage Was this helpful? Send feedback **Set up a Firebase Cloud Messaging client app on Android** FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store. **Set up the SDK** This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project  **Edit your app manifest** Add the following to your app's manifest: • A service that extends `FirebaseMessagingService`. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service. `<service android:name=".java.MyFirebaseMessagingService" android:exported="false"> <intent-filter> <action android:name="com.google.firebase.MESSAGING_EVENT" /> </intent-filter> </service>` AndroidManifest.xml

	 Access the device registration token On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`. This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token. The registration token may change when: • The app is restored on a new device • The user uninstalls/reinstall the app • The user clears app data. https://firebase.google.com/docs/cloud-messaging/android/client;



Firebase > Documentation > FCM > Engage

Was this helpful?

Receive messages in an Android app

Send feedback

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see Message types.

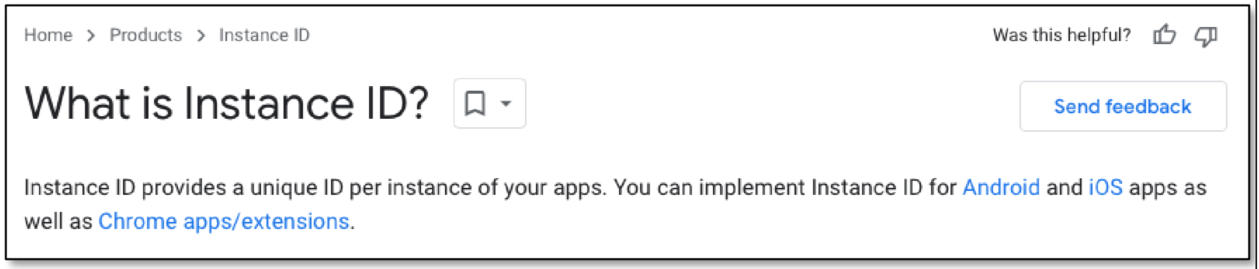
Handling messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's background execution limits may interfere with your ability to complete your work. For more information see our overview on message priority.

`onMessageReceived` is provided for most message types, with the following exceptions:

- **Notification messages delivered when your app is in the background**. In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

https://firebase.google.com/docs/cloud-messaging/android/receive;



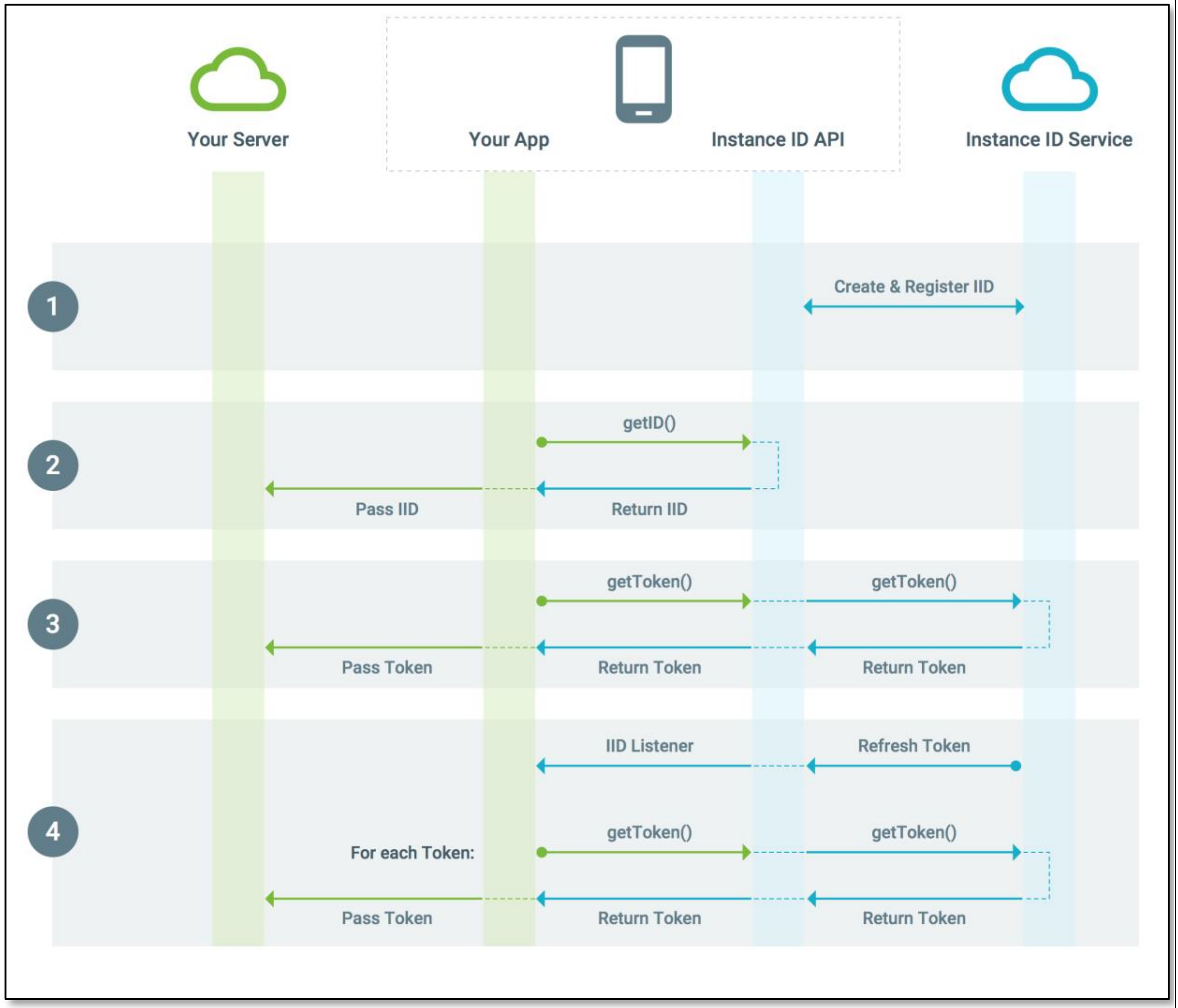
Home > Products > Instance ID

Was this helpful?

What is Instance ID?

Send feedback

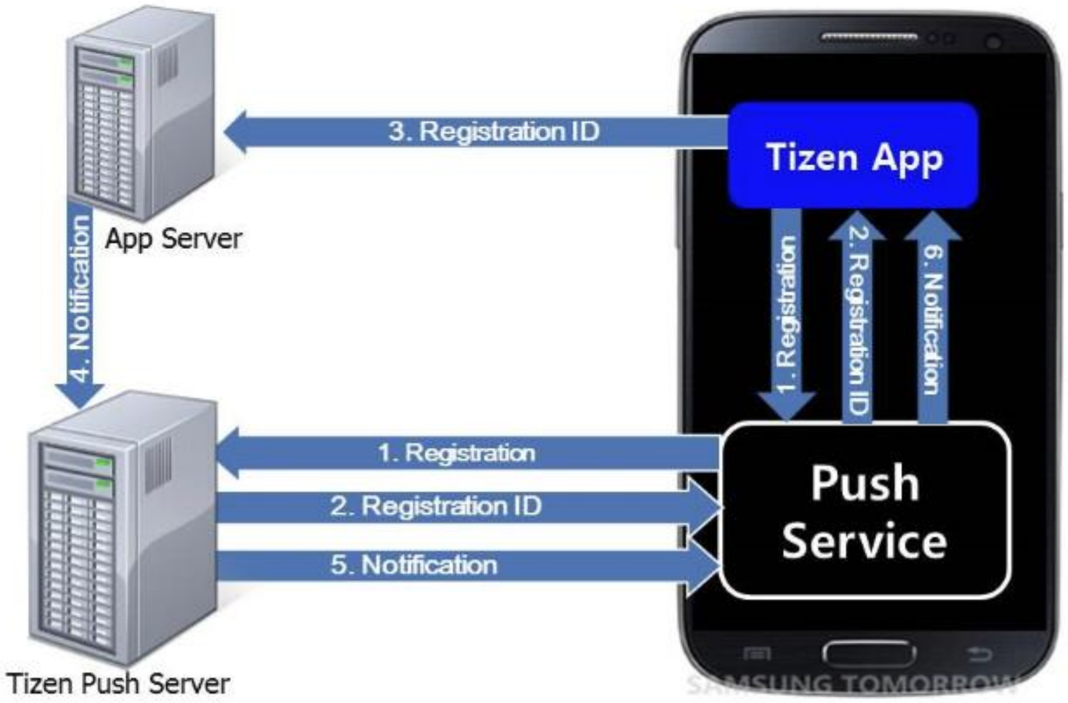
Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for Android and iOS apps as well as Chrome apps/extensions.

	 Your Server Your App Instance ID API Instance ID Service 1 Create & Register IID 2 getID() Pass IID Return IID 3 getToken() getToken() Pass Token Return Token Return Token 4 IID Listener Refresh Token For each Token: getToken() getToken() Pass Token Return Token Return Token https://developers.google.com/instance-id. For further example, Samsung's Tizen based devices comprise multiple device agents which have an identifier. *See e.g.*,

Architecture

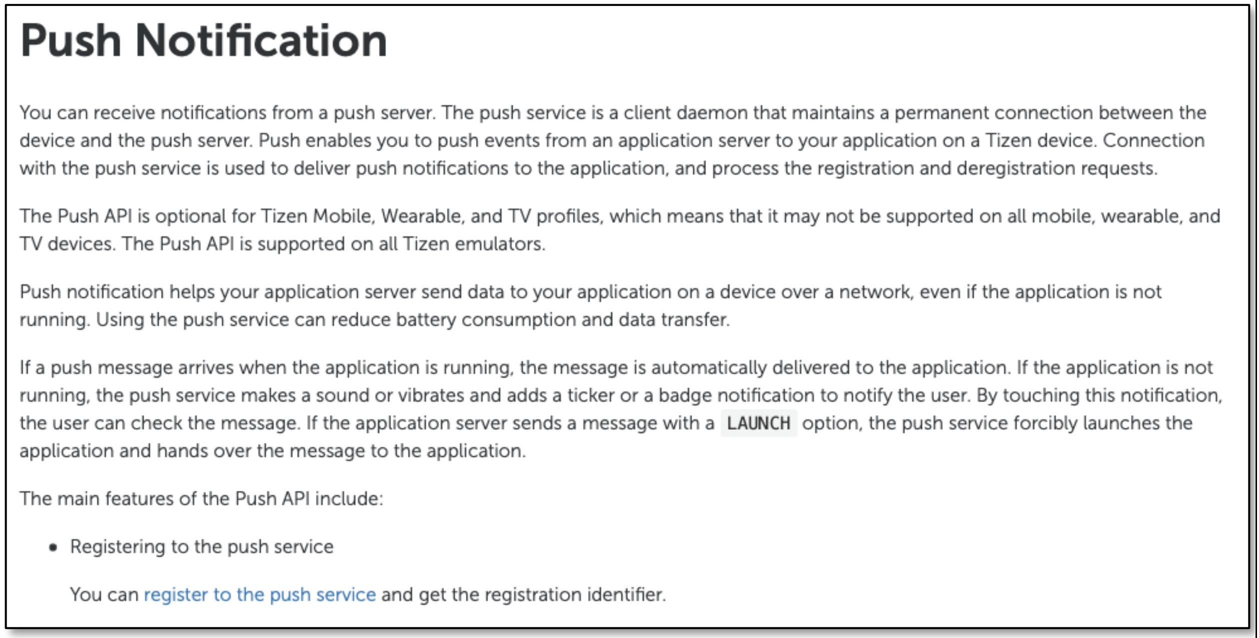
The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.



Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

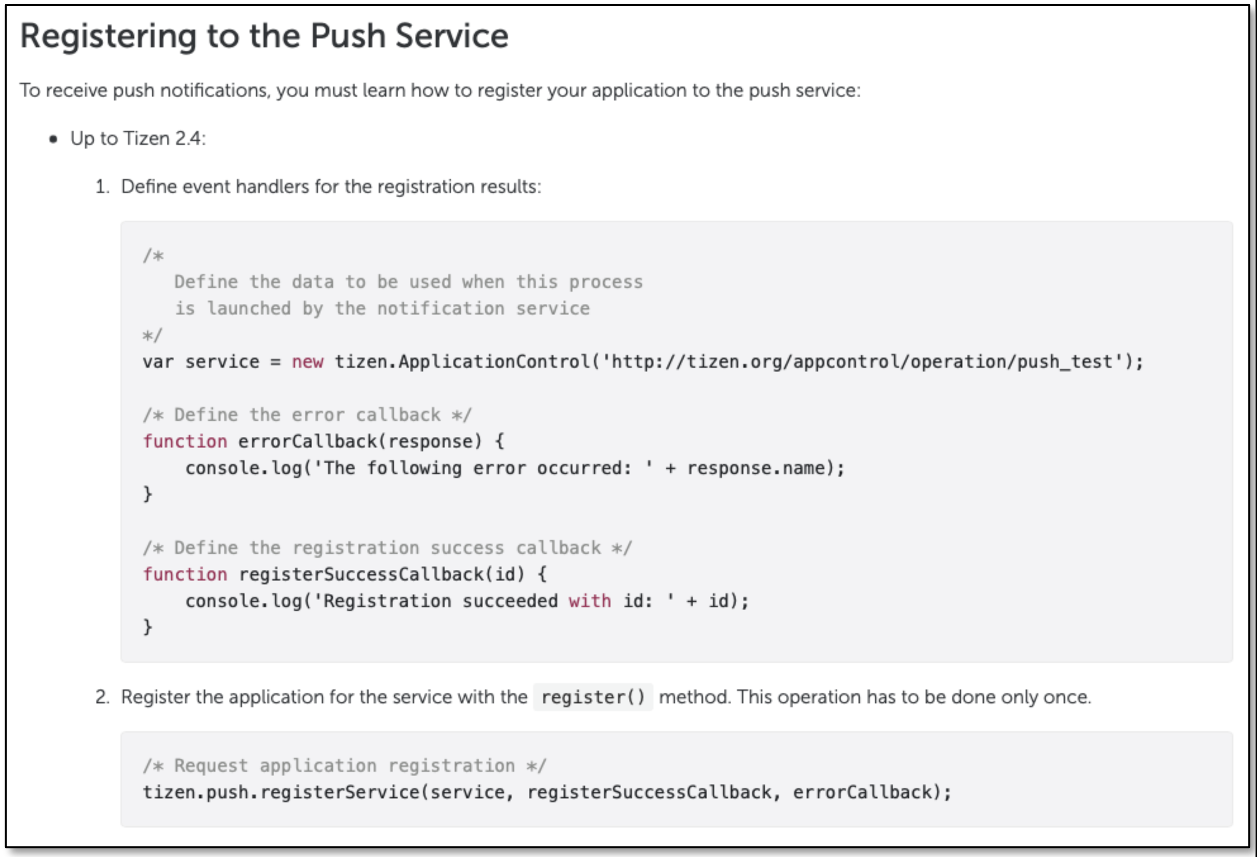
Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

  You can register to the push service and get the registration identifier.

Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

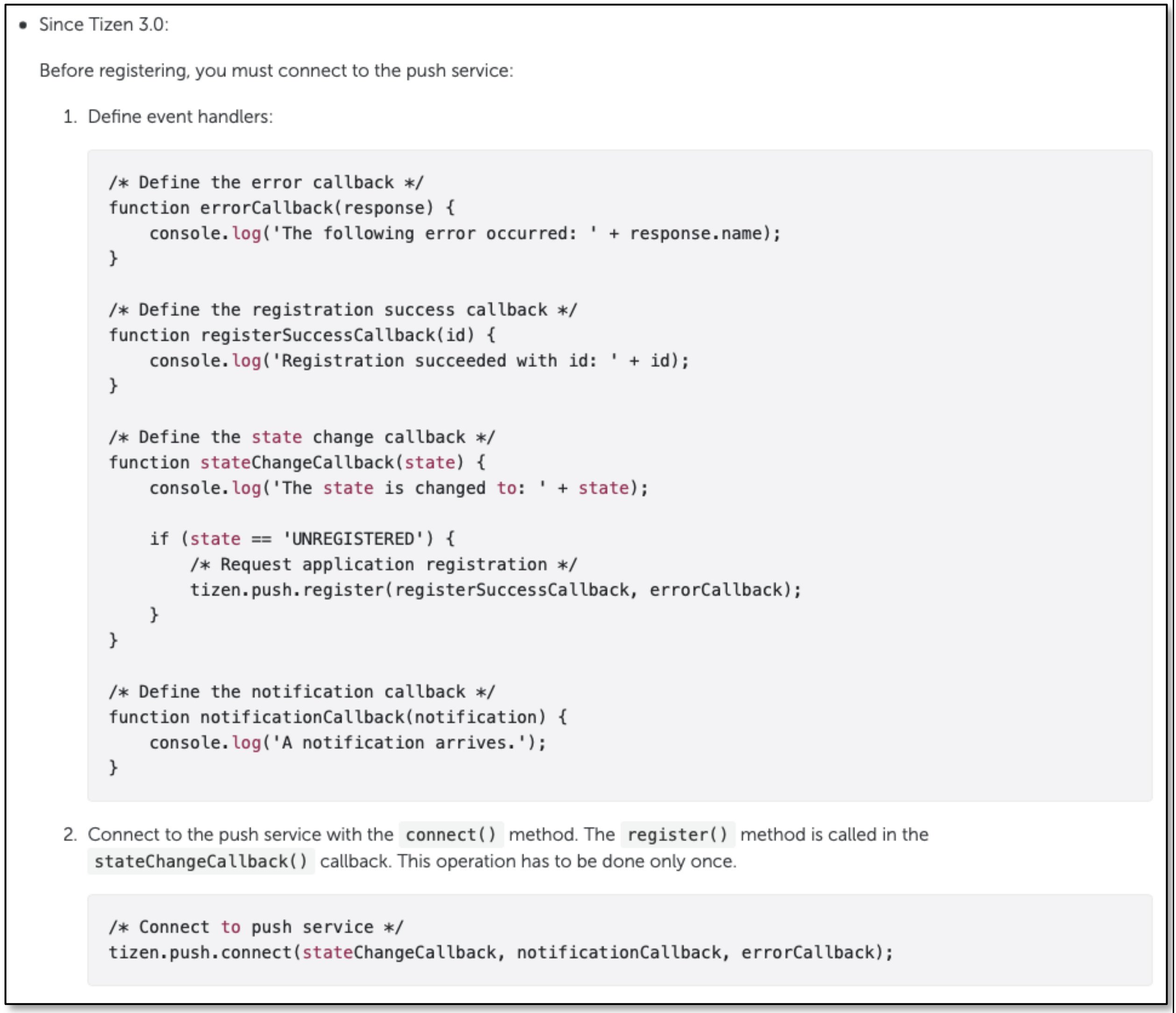
- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

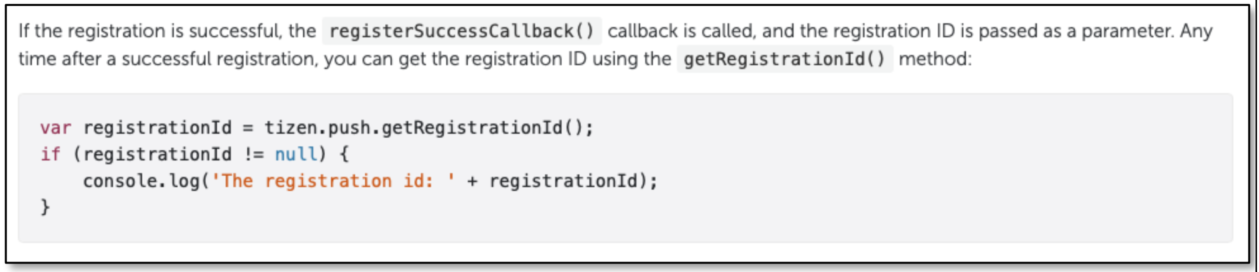
  Before registering, you must connect to the push service:

  1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

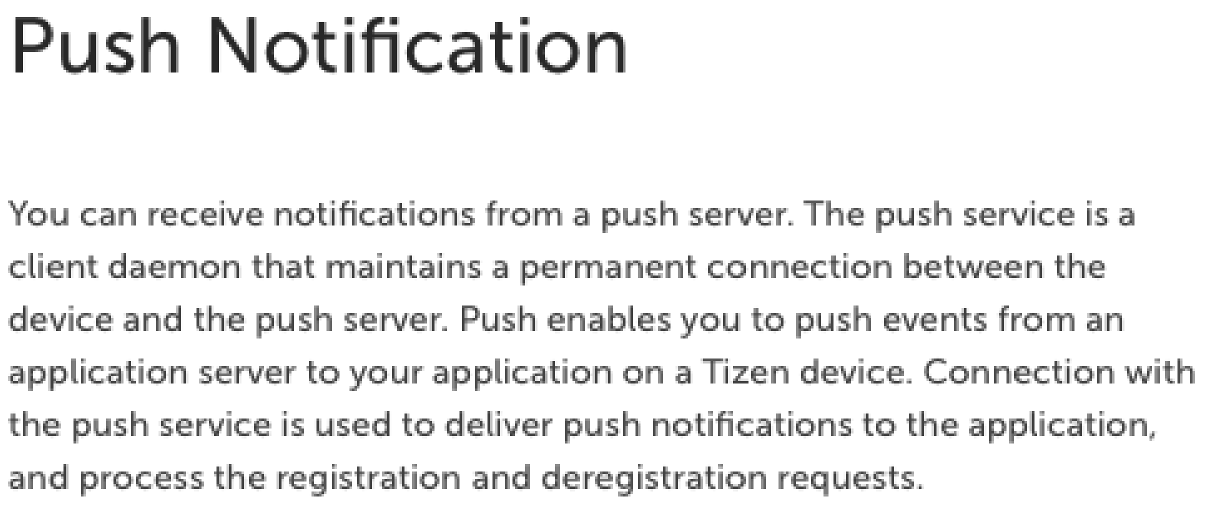
  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```


If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

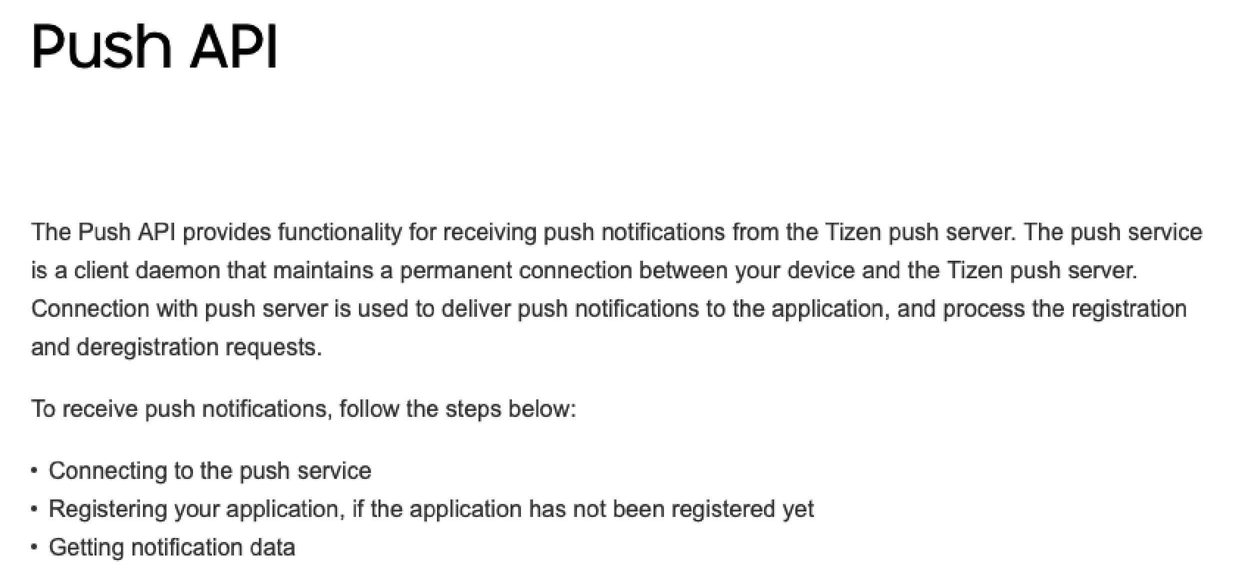
```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```

https://docs.tizen.org/application/web/guides/messaging/push/;


Push Notification

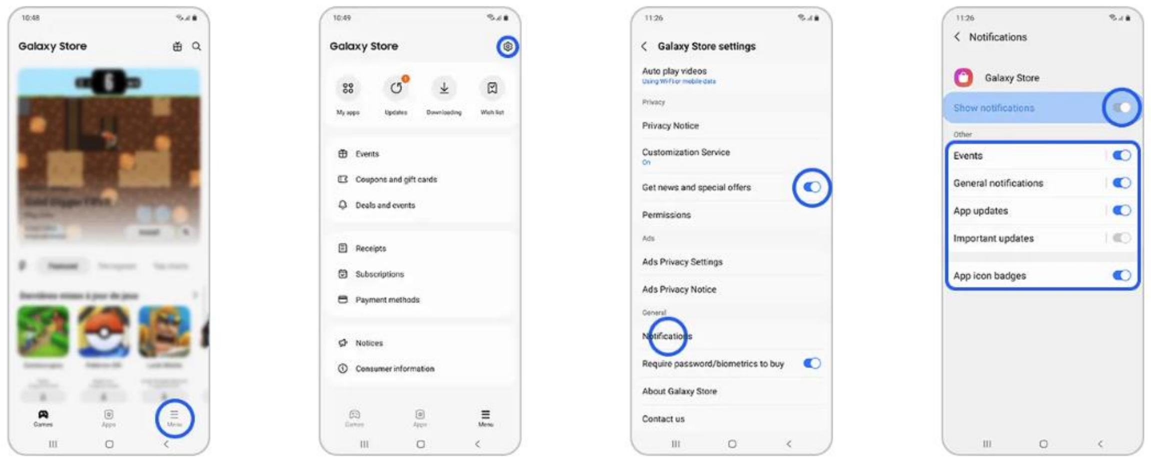
You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

https://developer.tizen.org/development/guides/web-application/messaging/push-notification;

	 Push API The Push API provides functionality for receiving push notifications from the Tizen push server. The push service is a client daemon that maintains a permanent connection between your device and the Tizen push server. Connection with push server is used to deliver push notifications to the application, and process the registration and deregistration requests. To receive push notifications, follow the steps below: • Connecting to the push service • Registering your application, if the application has not been registered yet • Getting notification data https://developer.samsung.com/smarttv/develop/api-references/tizen-web-device-api-references/push-api.html;

How to set up push notifications

If you want to set up push notifications, follow these steps.



Step 1. Log in to Galaxy Store and tap the "Menu" icon from the main screen.

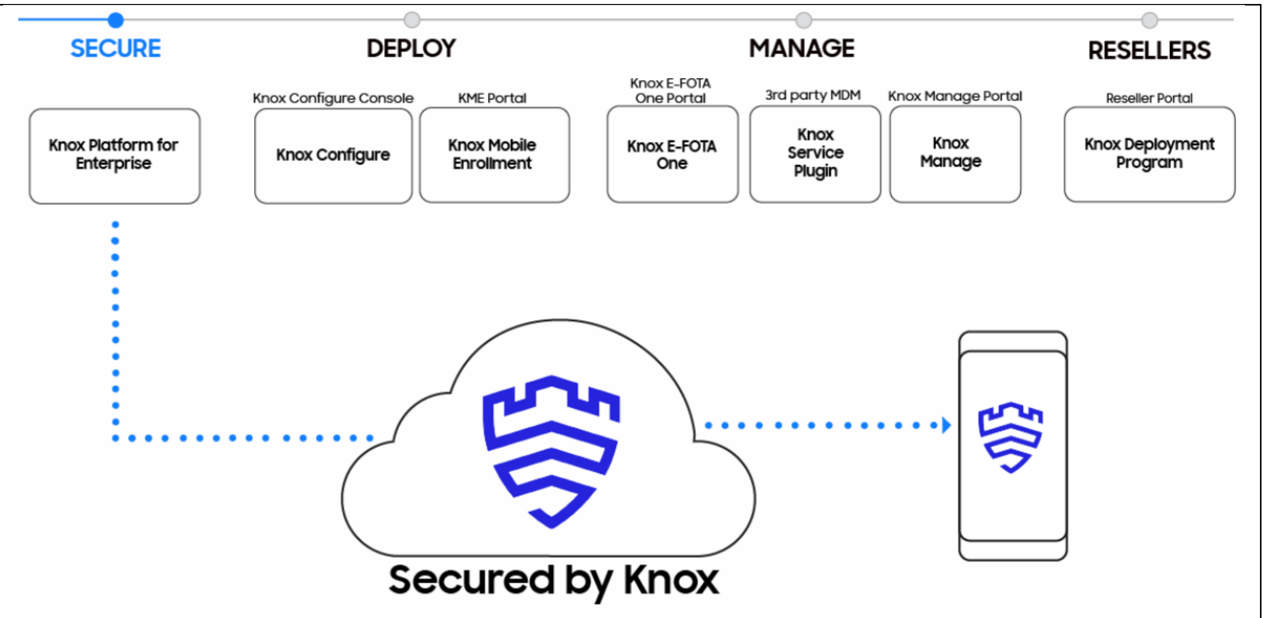
Step 2. Tap the settings icon located at the top right of the Galaxy Store menu.

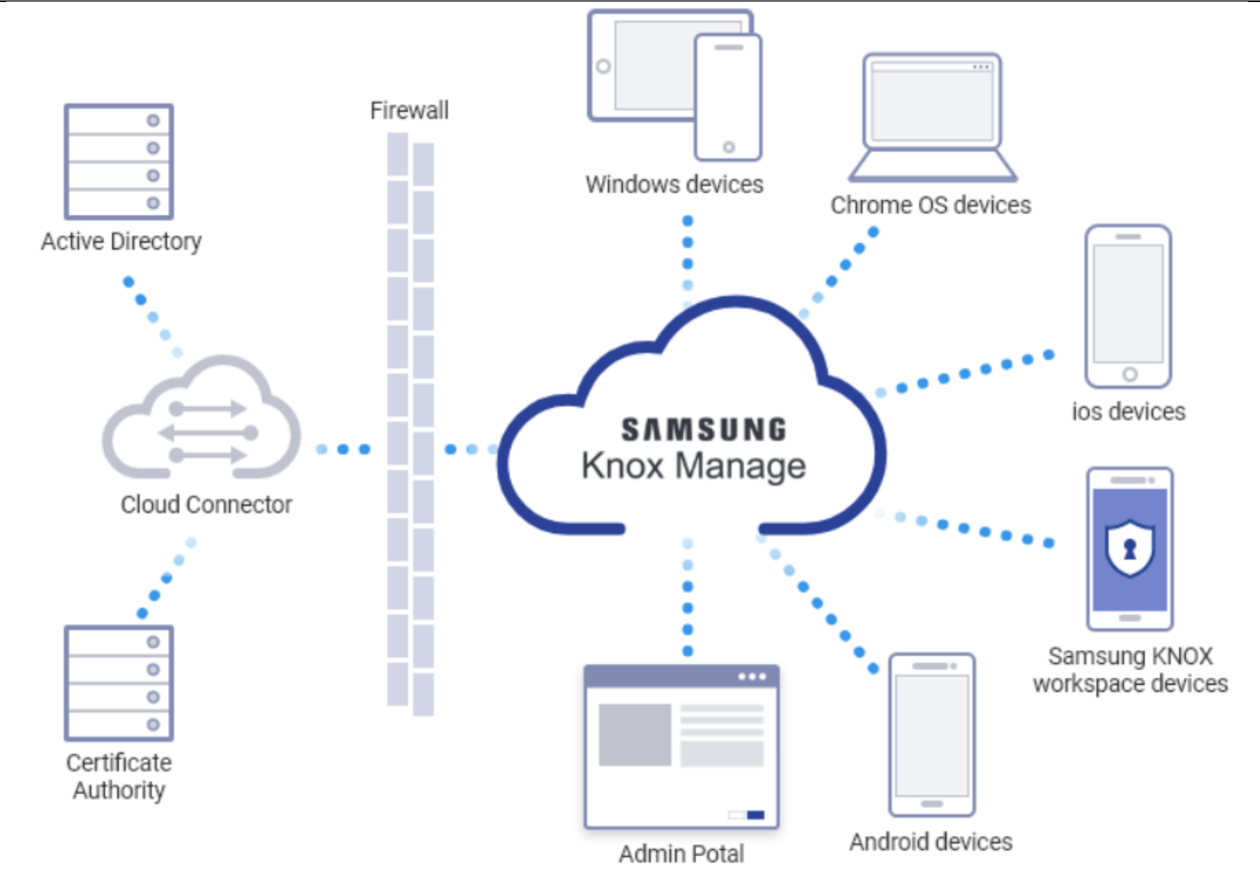
Step 3. Enable "Get news and special offers" and tap "Notifications."

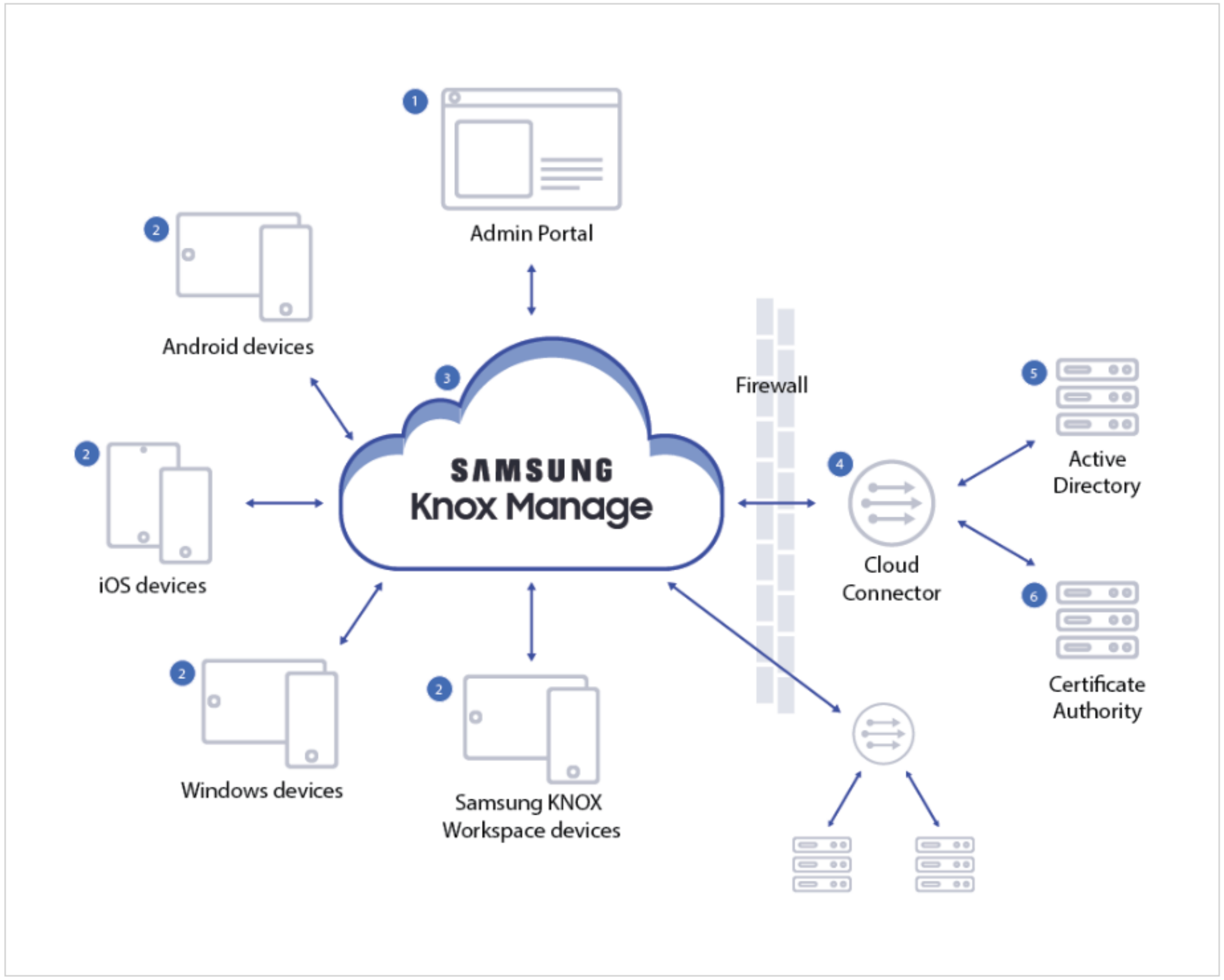
Step 4. Enable "Show notifications" and then enable the types of notifications you want to receive.

https://www.samsung.com/hk_en/support/apps-services/how-do-i-enable-push-notifications-from-galaxy-store/.

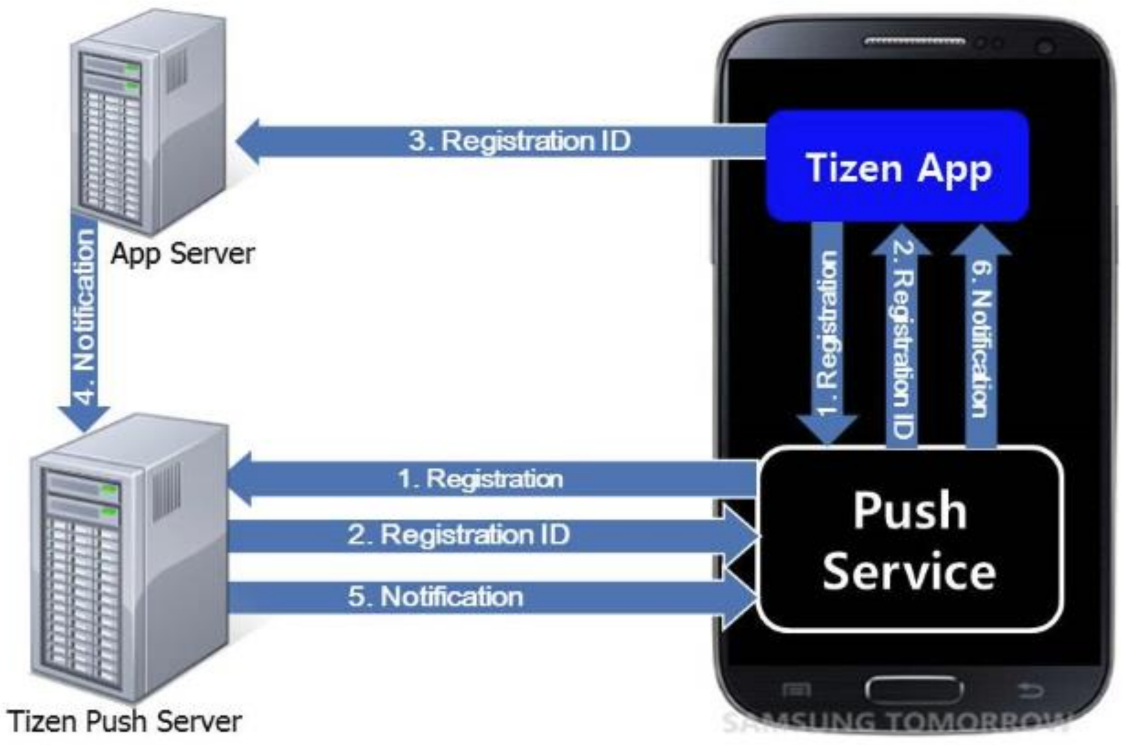
As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise multiple device agents which have an identifier.

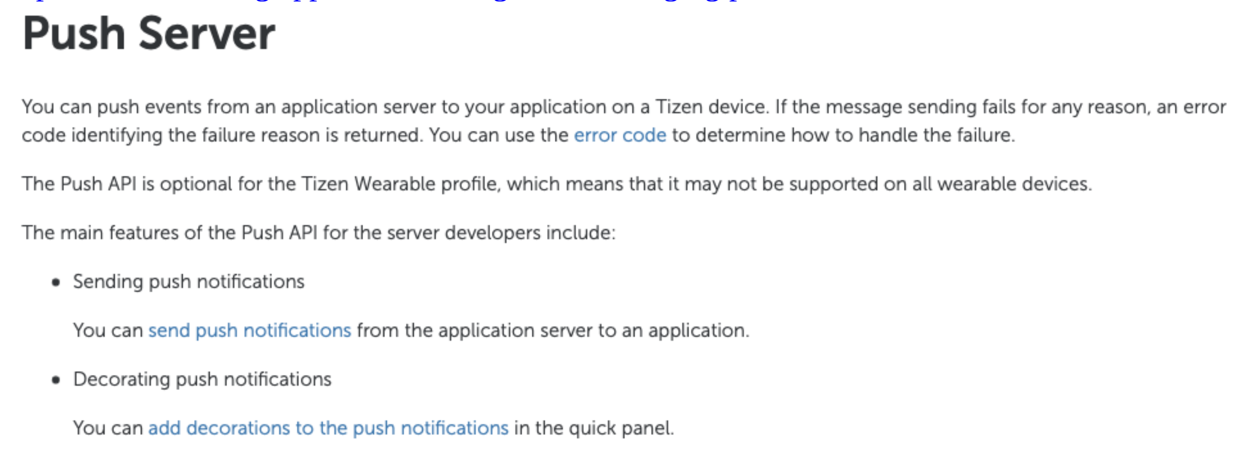
	 SECURE DEPLOY MANAGE RESELLERS Knox Configure Console KME Portal Knox E-FOTA One Portal 3rd party MDM Knox Manage Portal Reseller Portal Knox Platform for Enterprise Knox Configure Knox Mobile Enrollment Knox E-FOTA One Knox Service Plugin Knox Manage Knox Deployment Program Secured by Knox https://docs.samsungknox.com/admin/fundamentals/welcome.htm

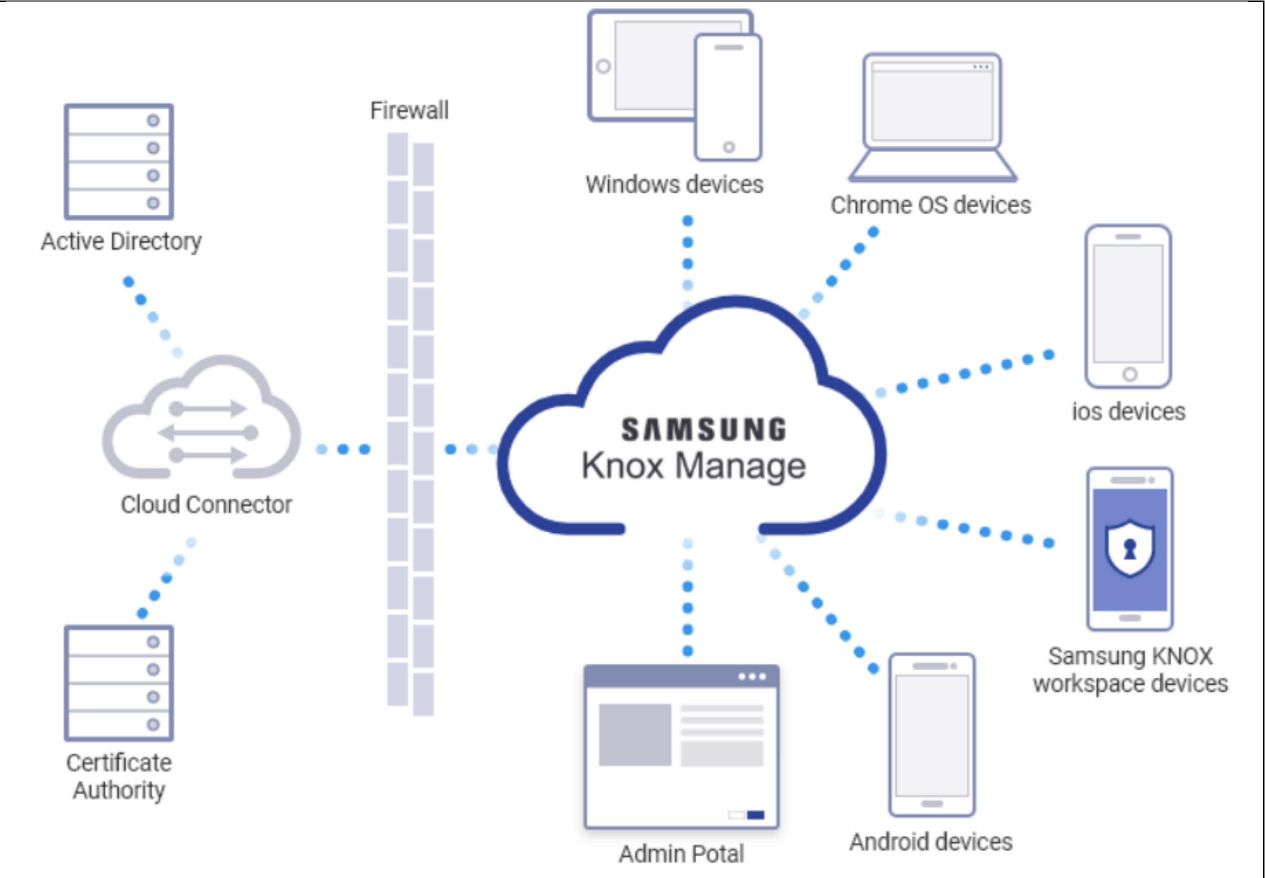
	 Active Directory Firewall Windows devices Chrome OS devices ios devices Cloud Connector SAMSUNG Knox Manage Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

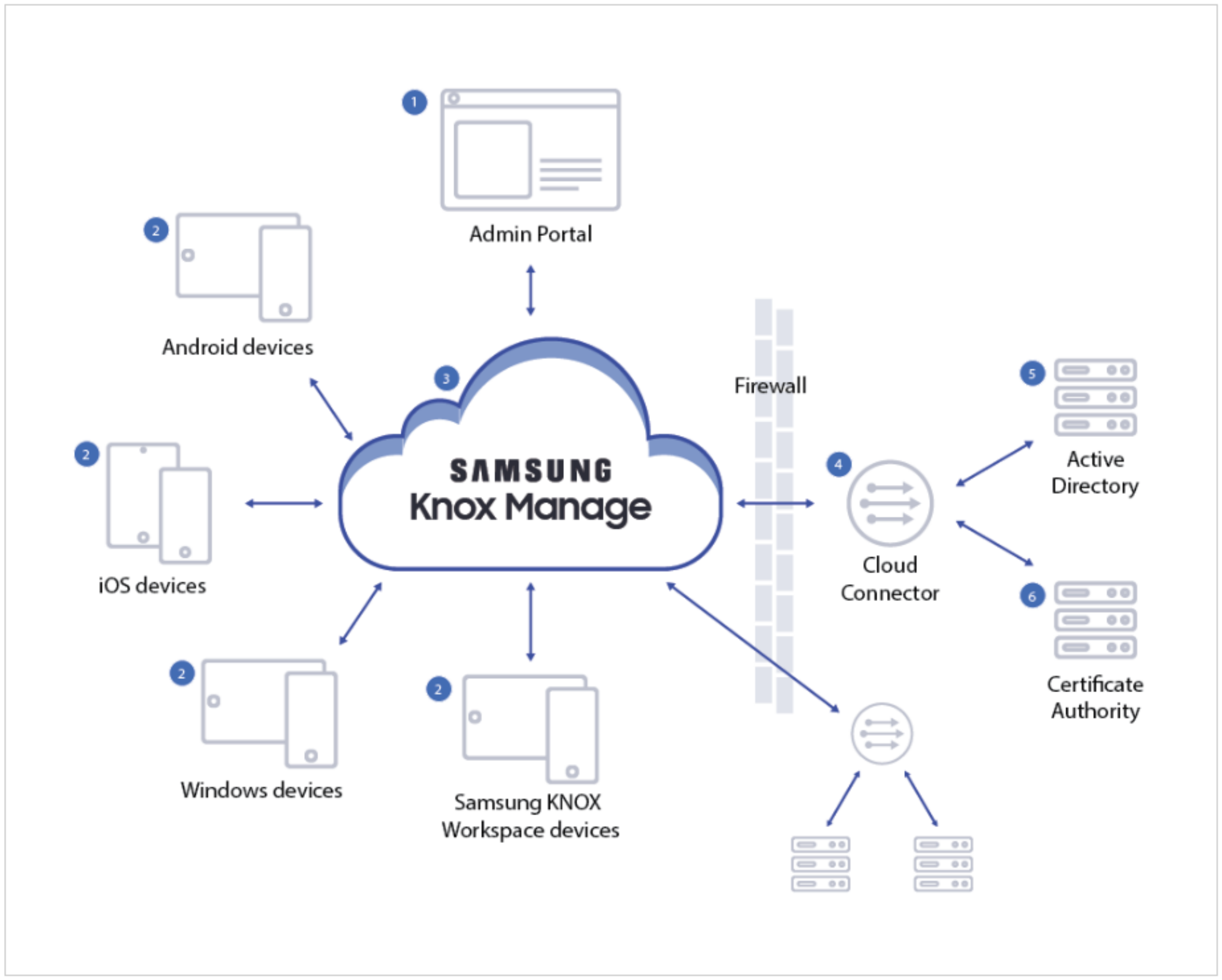
	 1 Admin Portal 2 Android devices 2 iOS devices 3 SAMSUNG Knox Manage Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority 2 Windows devices 2 Samsung KNOX Workspace devices https://docs.samsungknox.com/admin/knox-manage/km-features.htm

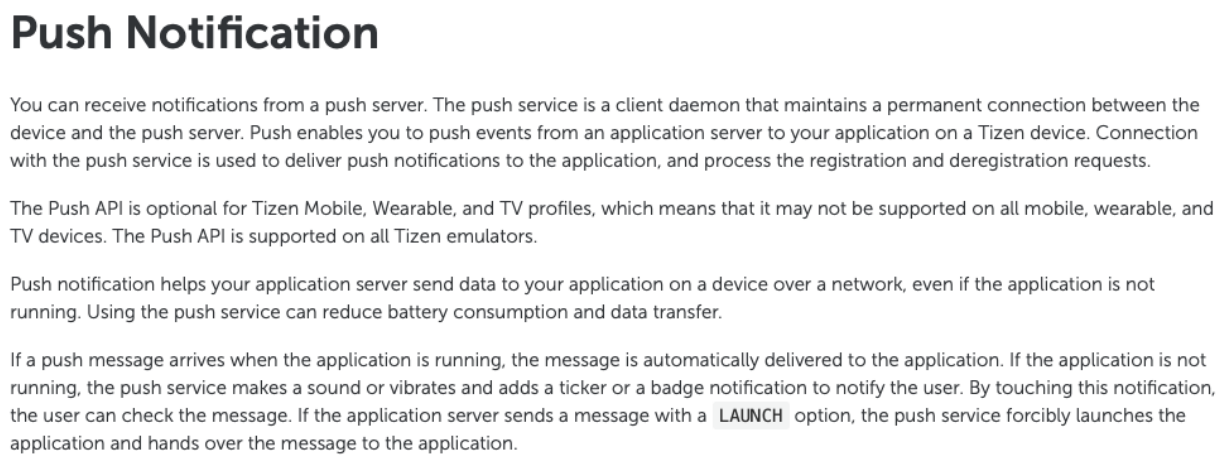
[1b] a network message server	Samsung devices and push messaging servers comprise "a network message server." For example, Samsung's push messaging servers are network servers. *See e.g.*,

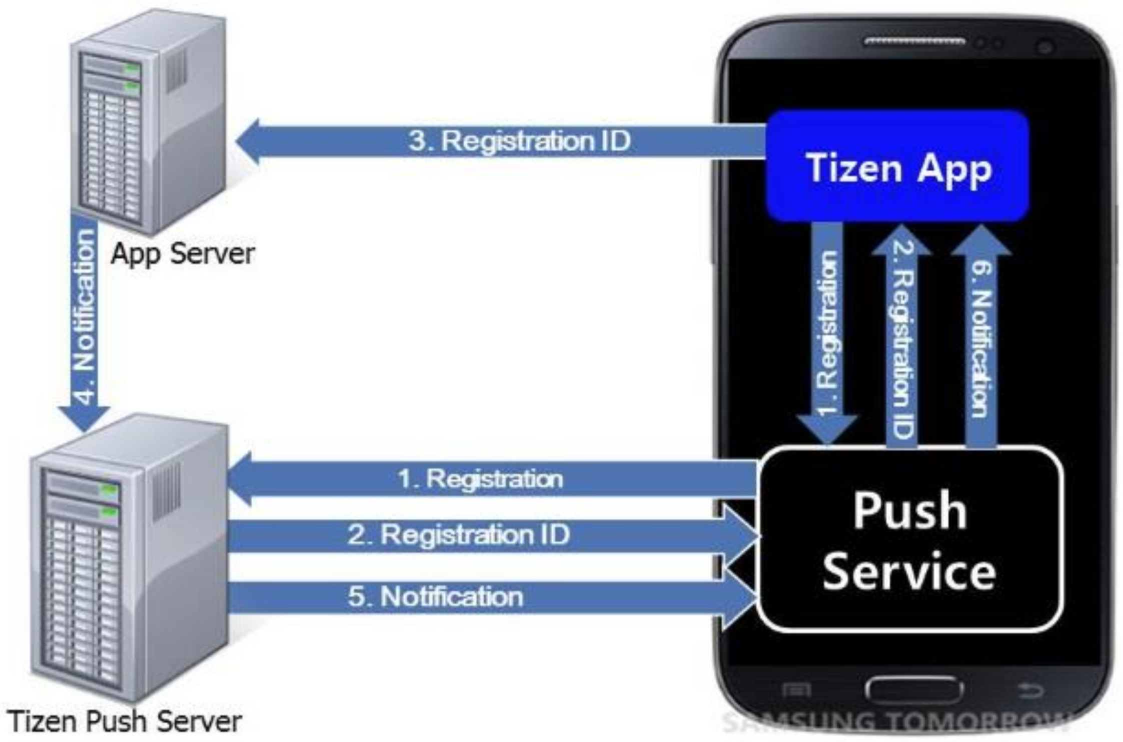
	Architecture The architecture of the Tizen Push service is described in detail in the mobile native Push guide. Figure: Service architecture  3. Registration ID Tizen App App Server 4. Notification 1. Registration 2. Registration ID 6. Notification 1. Registration 2. Registration ID 5. Notification Push Service Tizen Push Server SAMSUNG TOMORROW To receive push notifications for your application: 1. Request permission to access the Tizen push servers for using the push service API. 2. Wait for a confirmation email for the request. 3. Register the installed application on the device. 4. Connect to the push service for receiving push notifications. 5. Receive notifications from the push service.

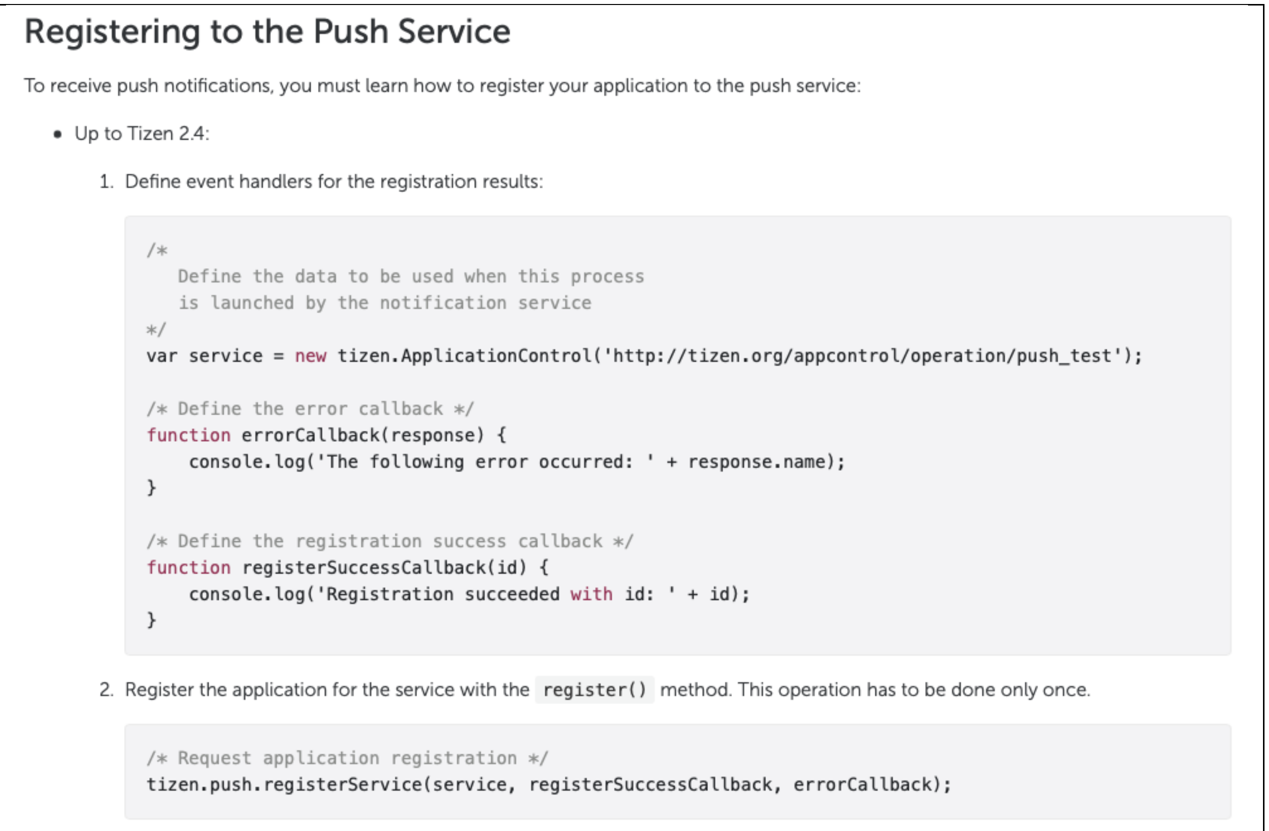
	https://docs.tizen.org/application/web/guides/messaging/push/;  **Push Server** You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. The main features of the Push API for the server developers include: • Sending push notifications — You can send push notifications from the application server to an application. • Decorating push notifications — You can add decorations to the push notifications in the quick panel. https://docs.tizen.org/application/native/guides/messaging/push-server/; As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, include network message servers.

	 Active Directory Firewall Windows devices Chrome OS devices Cloud Connector SAMSUNG Knox Manage ios devices Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 1 Admin Portal 2 Android devices 2 iOS devices 3 SAMSUNG Knox Manage Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority 2 Windows devices 2 Samsung KNOX Workspace devices https://docs.samsungknox.com/admin/knox-manage/km-features.htm
[1c] supporting a plurality of secure Internet data connections, each secure Internet data connection between the network message server and a respective one of the mobile end-user devices via a	Samsung's push messaging servers "support[] a plurality of secure Internet data connections, each secure Internet data connection between the network message server and a respective one of the mobile end-user devices via a device data connection to a wireless network."

device data connection to a wireless network,	For example, Samsung's push messaging server supports a plurality of secure connections between the network message server and respective devices. *See, e.g.,*  **Push Notification** You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests. The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators. Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

	## Architecture The architecture of the Tizen Push service is described in detail in the mobile native Push guide. Figure: Service architecture  App Server 3. Registration ID Tizen App 4. Notification 1. Registration 2. Registration ID 6. Notification 1. Registration 2. Registration ID 5. Notification Push Service Tizen Push Server SAMSUNG TOMORROW To receive push notifications for your application: 1. Request permission to access the Tizen push servers for using the push service API. 2. Wait for a confirmation email for the request. 3. Register the installed application on the device. 4. Connect to the push service for receiving push notifications. 5. Receive notifications from the push service.



Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

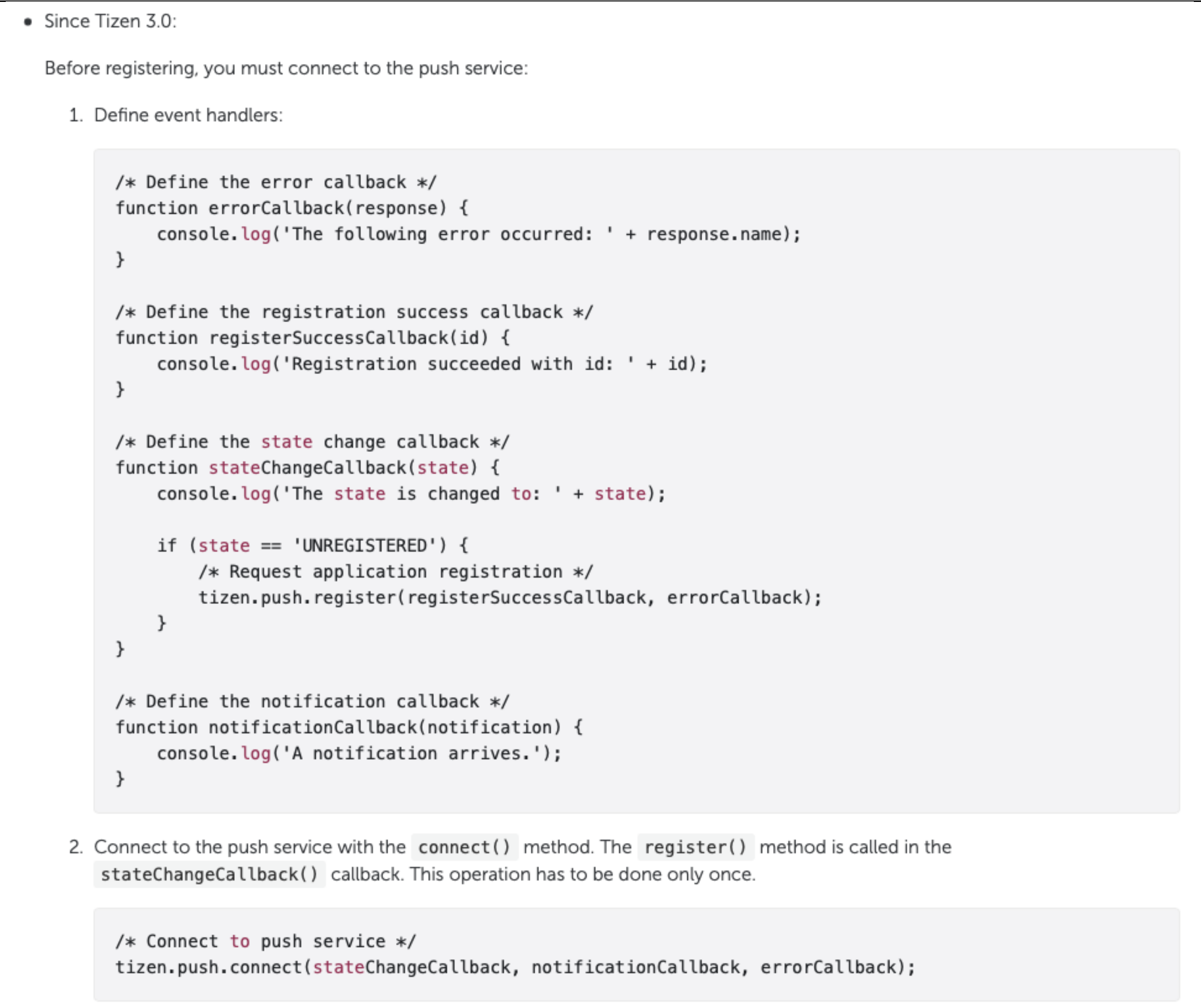
- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

  Before registering, you must connect to the push service:

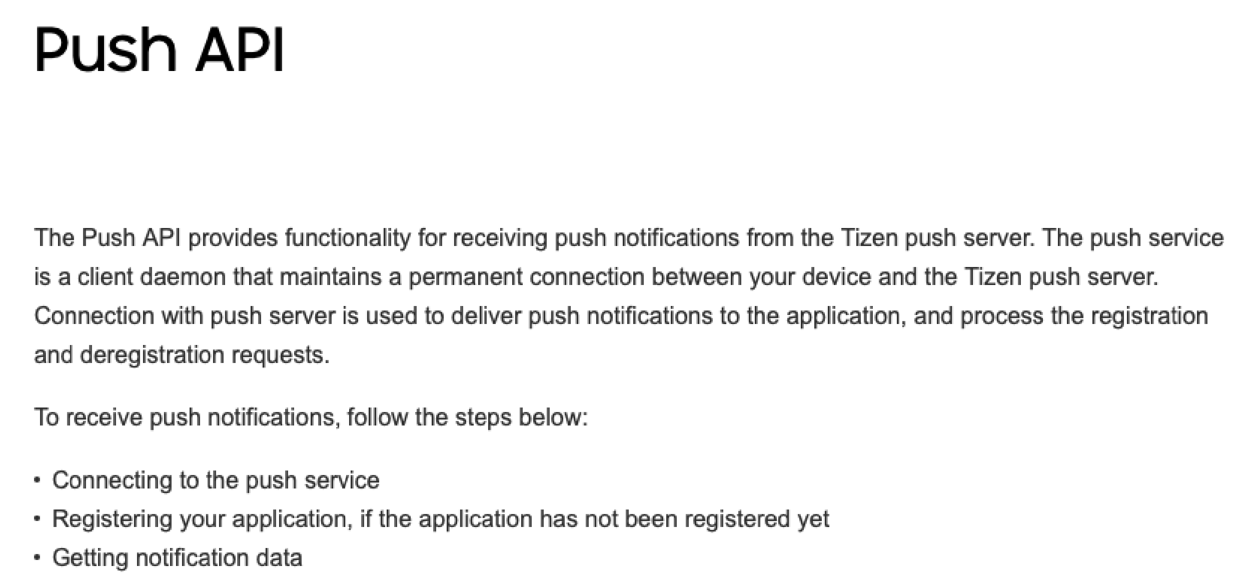
  1. Define event handlers:

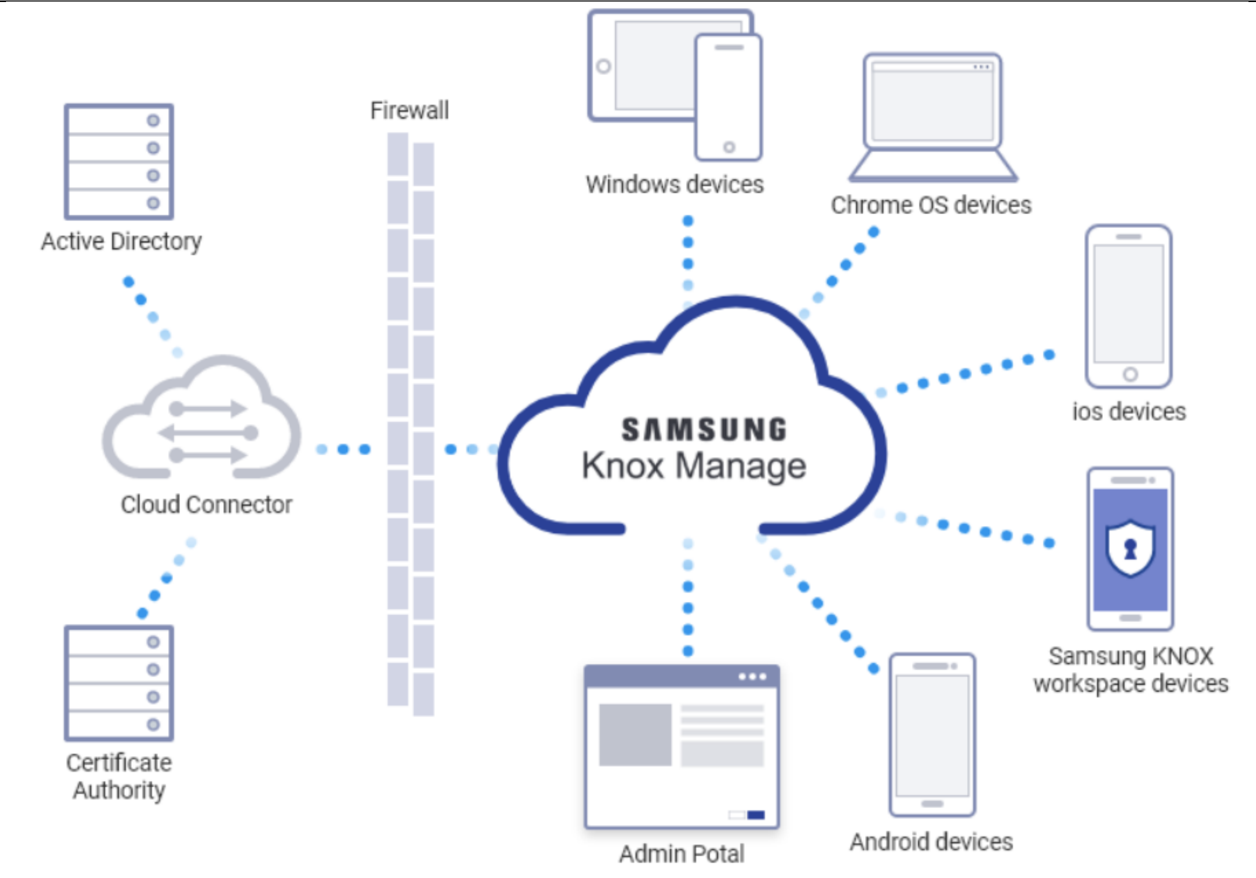
```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

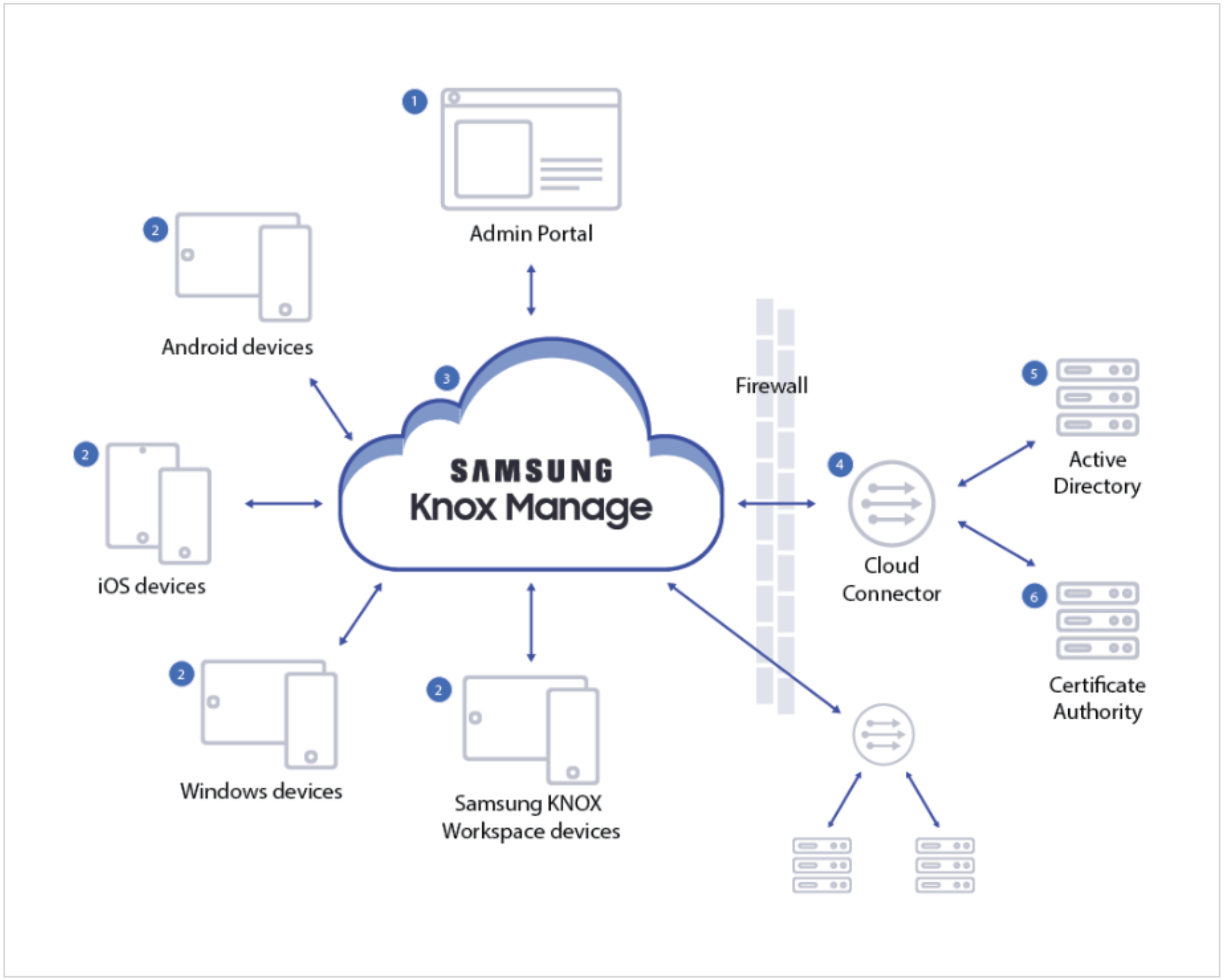
  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

https://docs.tizen.org/application/web/guides/messaging/push/;

	 Push API The Push API provides functionality for receiving push notifications from the Tizen push server. The push service is a client daemon that maintains a permanent connection between your device and the Tizen push server. Connection with push server is used to deliver push notifications to the application, and process the registration and deregistration requests. To receive push notifications, follow the steps below: • Connecting to the push service • Registering your application, if the application has not been registered yet • Getting notification data https://developer.samsung.com/smarttv/develop/api-references/tizen-web-device-api-references/push-api.html. As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a plurality of secure Internet data connections (between managed devices and Samsung Knox servers).

	 Active Directory Firewall Windows devices Chrome OS devices ios devices Cloud Connector SAMSUNG Knox Manage Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

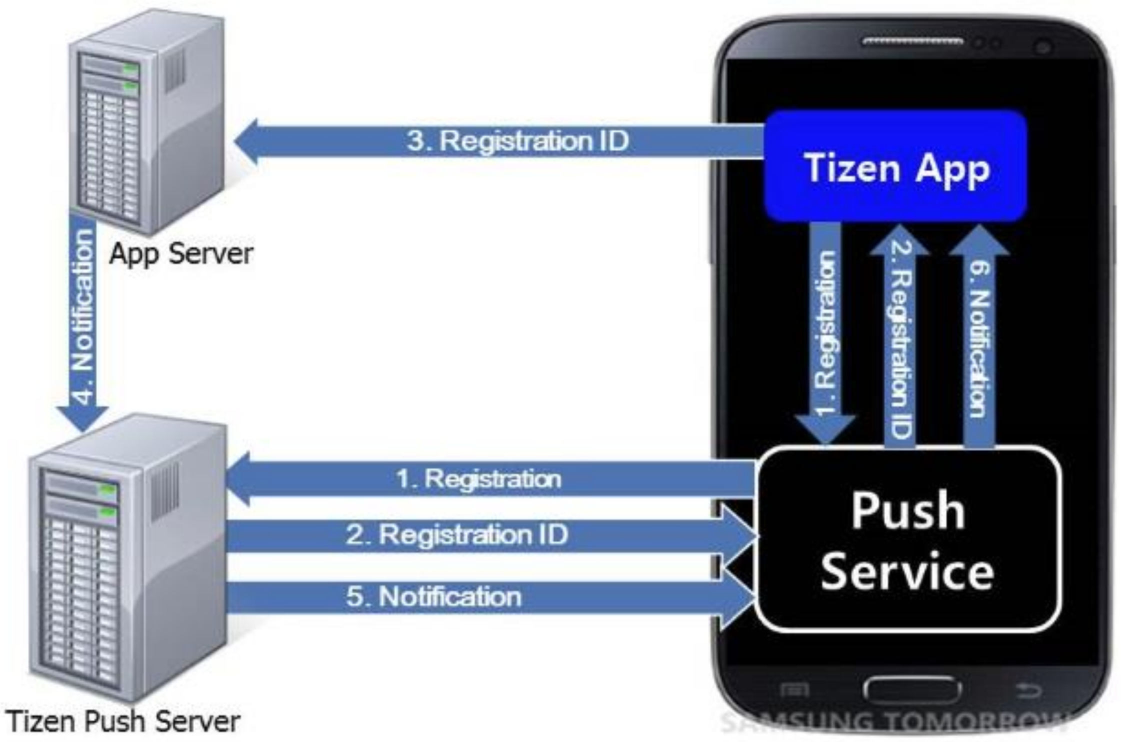
	 1 Admin Portal 2 Android devices 3 SAMSUNG Knox Manage 2 iOS devices Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority 2 Windows devices 2 Samsung KNOX Workspace devices https://docs.samsungknox.com/admin/knox-manage/km-features.htm As a further example, the communications links between Samsung Knox Manage servers or Knox MDM platform and devices being managed by such servers or MDM platform, for example including the Knox workspace devices, Android devices, iOS devices, Chrome OS devices, and Windows devices illustrated above, comprise secure Internet data connections, in that the connections are secure (insofar as they are encrypted per Samsung's requirements) and used to transmit information over the Internet. As a further example, the communications link

	between Samsung's Tizen servers and Tizen OS devices, and Firebase messaging servers and Samsung's Android devices, comprises a secure Internet data connections. Like with commands and messages sent between Knox servers and Knox-managed devices, notifications, data, and messages transmitted over these communications links in the Tizen and Firebase messaging framework may be encrypted, as illustrated above. See, e.g., (https://docs.tizen.org/application/native/guides/messaging/push/#security ("ret = push_service_get_notification_data(noti, &data); /* Decrypt app data here if it is encrypted */"))
[1d] the network message server configured to receive, from each of a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user devices and one of a plurality of applications,	Samsung's push messaging servers comprise "the network message server configured to receive, from each of a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user devices and one of a plurality of applications." For example, Samsung's push messaging servers receive messages from a plurality of network elements (e.g., App Servers) multiple requests to transmit data indicating a device and application. *See, e.g.*,

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

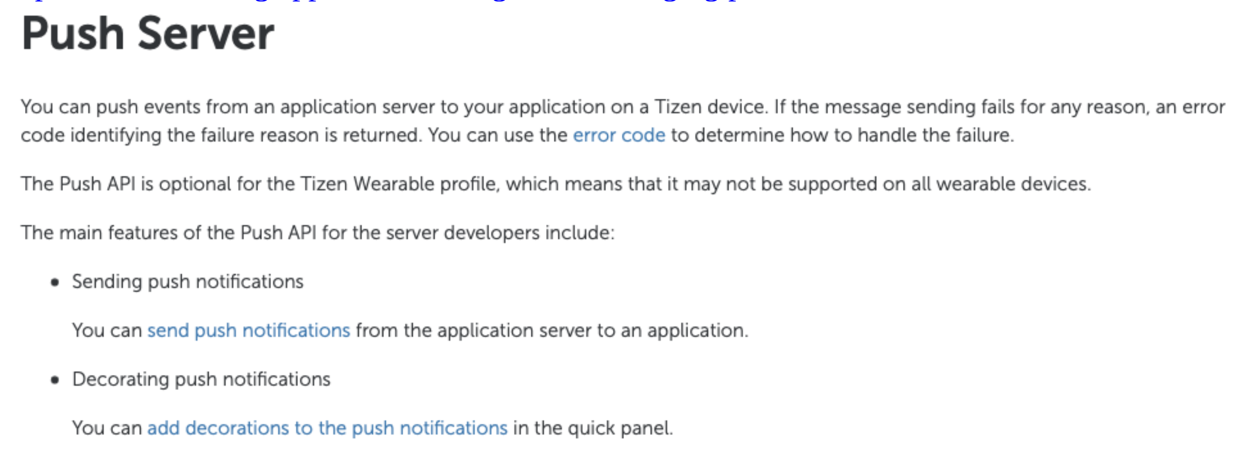
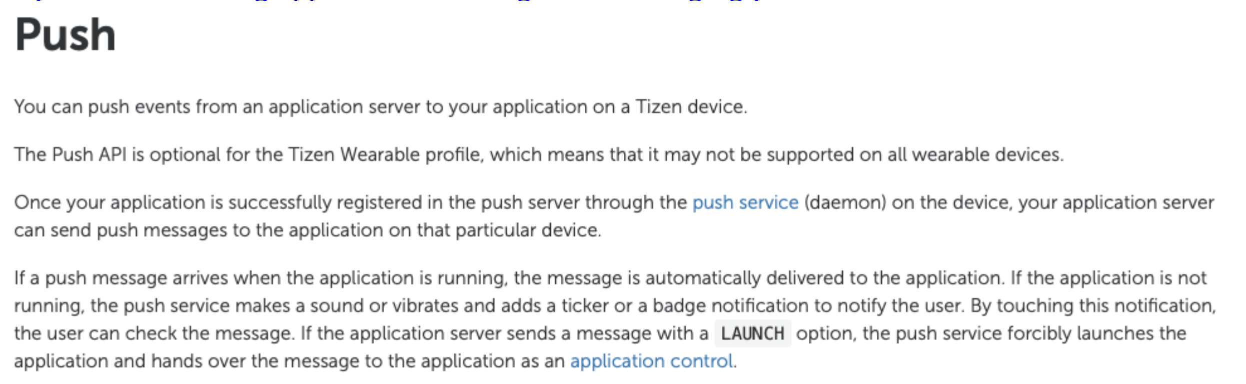
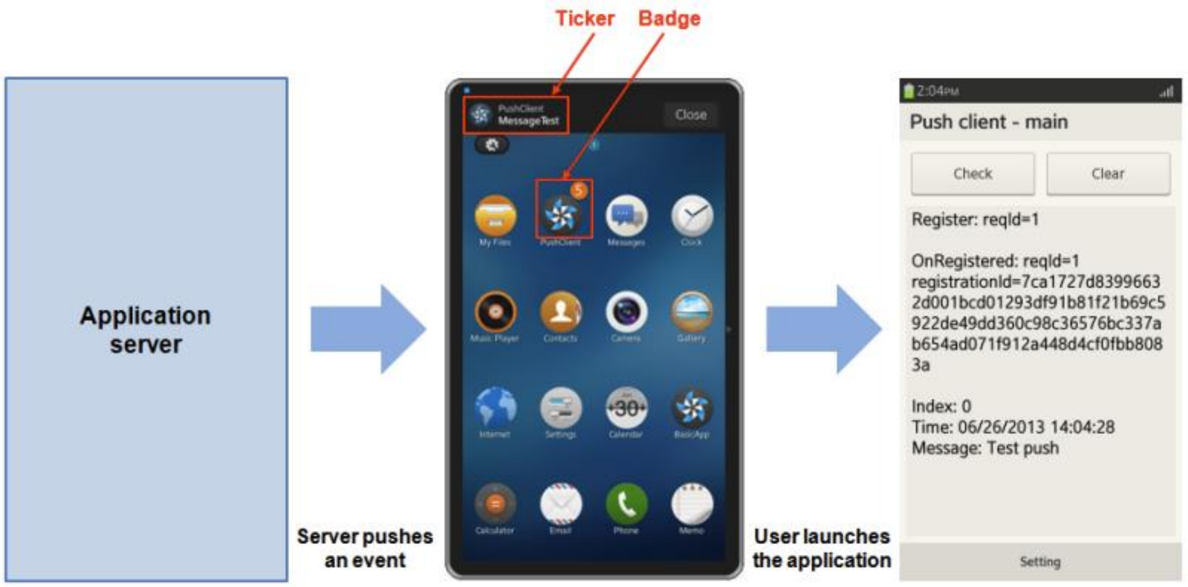
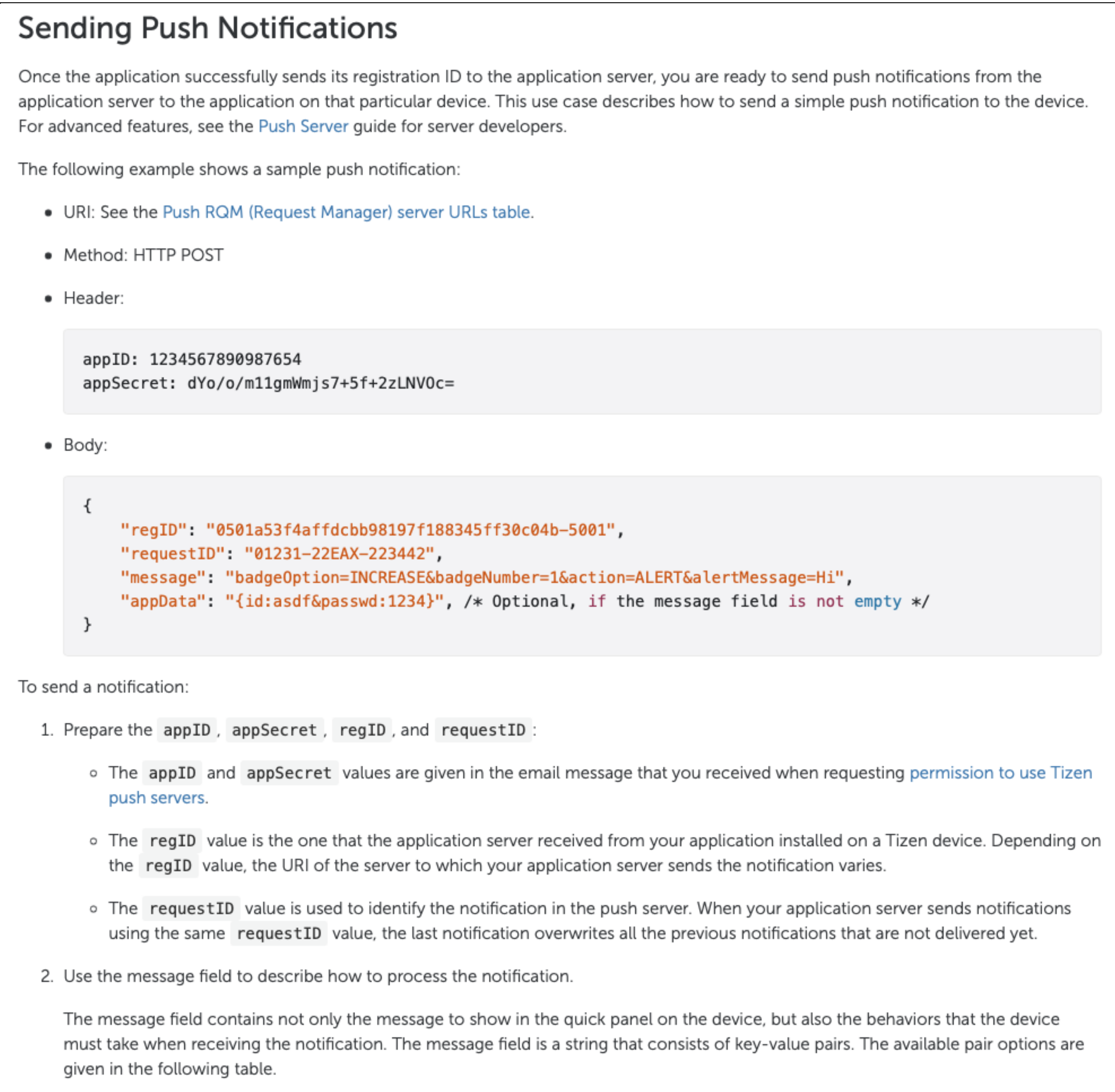
	https://docs.tizen.org/application/web/guides/messaging/push/;  **Push Server** You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. The main features of the Push API for the server developers include: • Sending push notifications You can send push notifications from the application server to an application. • Decorating push notifications You can add decorations to the push notifications in the quick panel. https://docs.tizen.org/application/native/guides/messaging/push-server/;  **Push** You can push events from an application server to your application on a Tizen device. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. Once your application is successfully registered in the push server through the push service (daemon) on the device, your application server can send push messages to the application on that particular device. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application as an application control.

	Figure: Push messaging service  Ticker Badge Application server Server pushes an event User launches the application PushClient MessageTest Close 2:04PM Push client - main Check Clear Setting Register: reqId=1 OnRegistered: reqId=1 registrationId=7ca1727d8399663 2d001bcd01293df91b81f21b69c5 922de49dd360c98c36576bc337a b654ad071f912a448d4cf0fbb808 3a Index: 0 Time: 06/26/2013 14:04:28 Message: Test push



Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

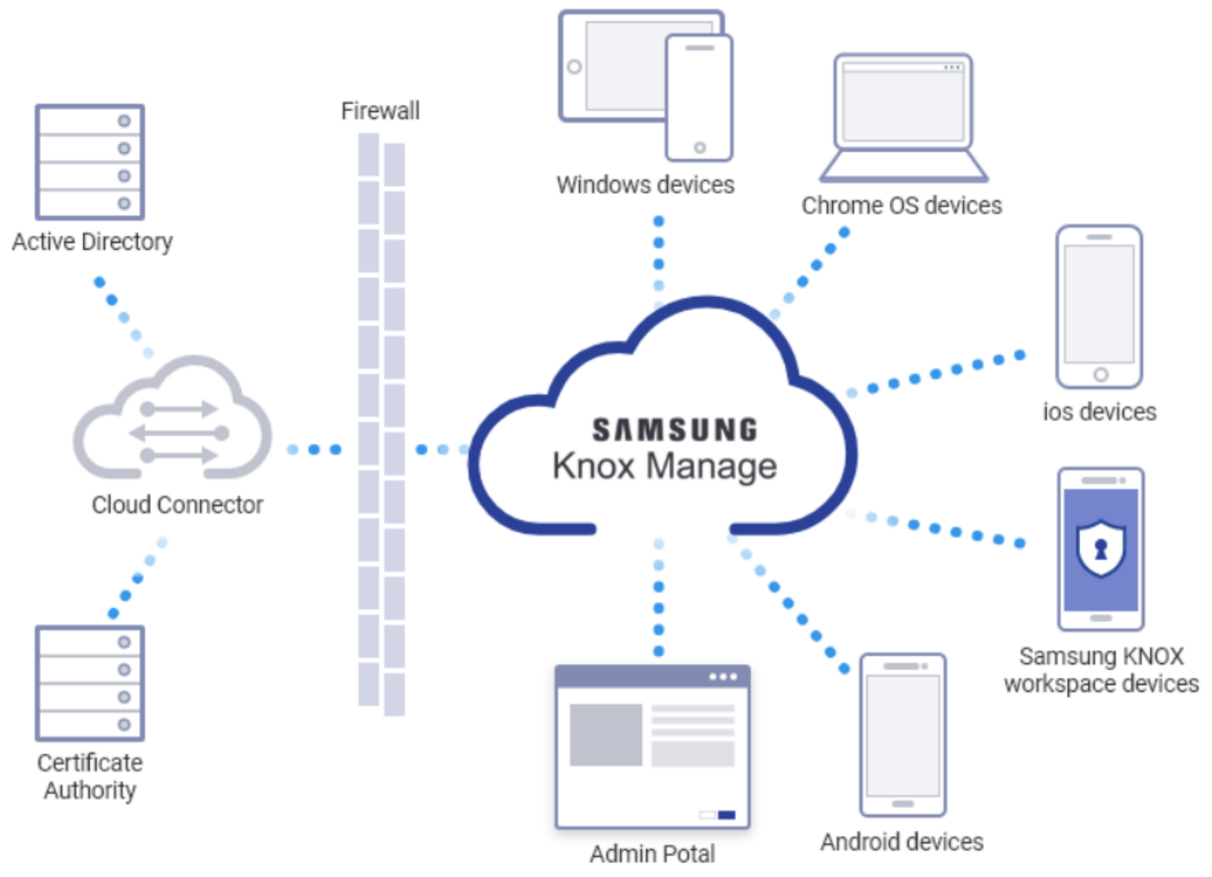
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

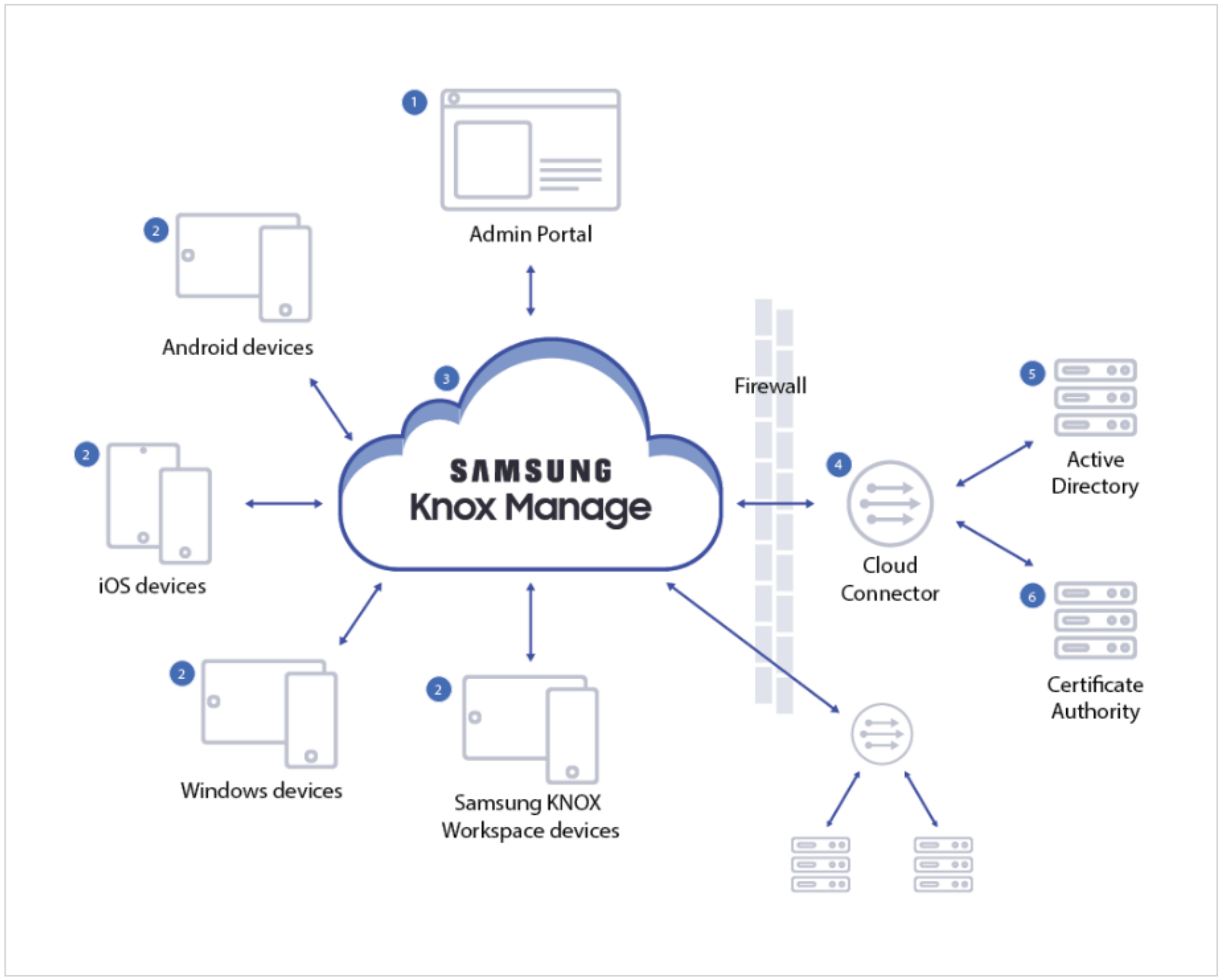
To send a notification:

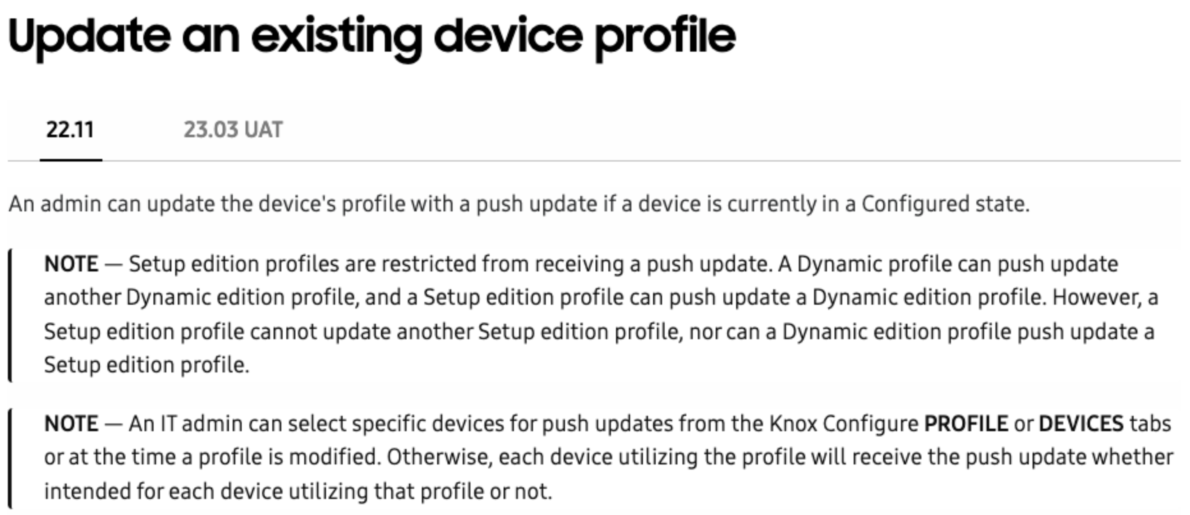
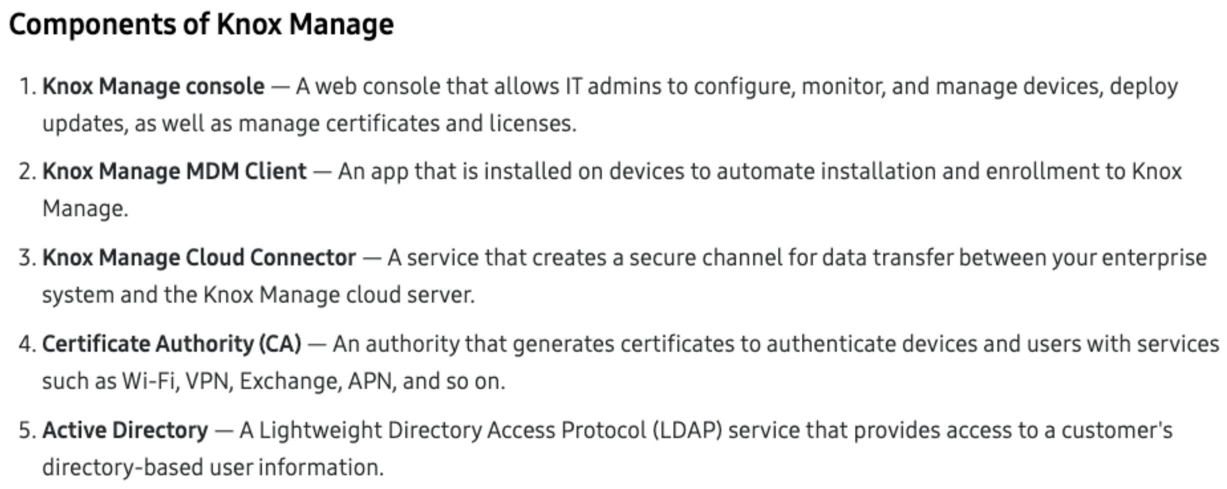
1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
   - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
   - The `regID` value is the one that the application server received from your application installed on a Tizen device. Depending on the `regID` value, the URI of the server to which your application server sends the notification varies.
   - The `requestID` value is used to identify the notification in the push server. When your application server sends notifications using the same `requestID` value, the last notification overwrites all the previous notifications that are not delivered yet.
2. Use the message field to describe how to process the notification.

   The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.

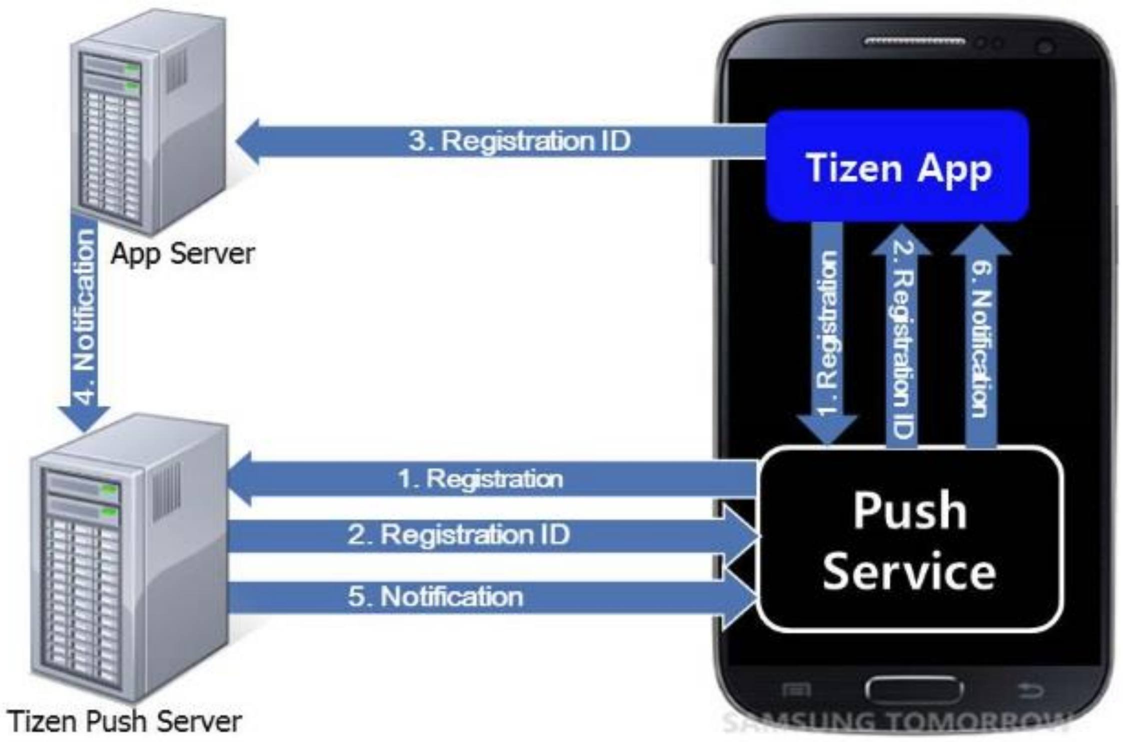
https://docs.tizen.org/application/native/guides/messaging/push/.

	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user devices and one of a plurality of applications.  Firewall Windows devices Chrome OS devices Active Directory ios devices SAMSUNG Knox Manage Cloud Connector Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

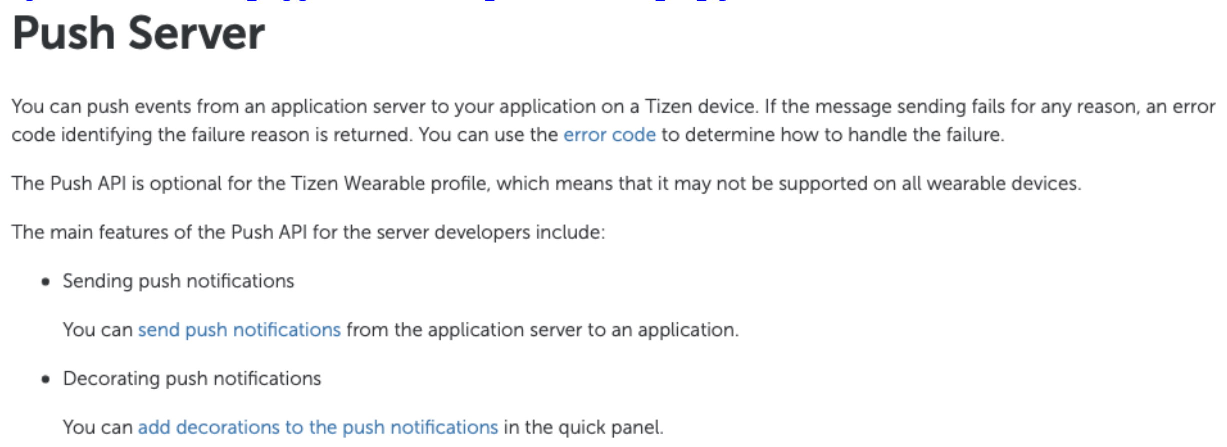
	 1 Admin Portal 2 Android devices 2 iOS devices 3 SAMSUNG Knox Manage Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority 2 Windows devices 2 Samsung KNOX Workspace devices https://docs.samsungknox.com/admin/knox-manage/km-features.htm

	 **Update an existing device profile** 22.11 23.03 UAT An admin can update the device's profile with a push update if a device is currently in a Configured state. **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile. **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not. https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm  **Components of Knox Manage** 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
[1e] the network message server to generate corresponding Internet data messages based on the requests, each	Samsung's push messaging servers comprise "the network message server to generate corresponding Internet data messages based on the requests, each such message containing at

such message containing at least one application identifier for an indicated application and application data corresponding to one of the requests, and	least one application identifier for an indicated application and application data corresponding to one of the requests." For example, Samsung's push messaging servers generate data messages containing an application identifier and data corresponding to the requests. *See*, *e.g.*,

	Architecture The architecture of the Tizen Push service is described in detail in the mobile native Push guide. **Figure: Service architecture**  App Server 3. Registration ID Tizen App 4. Notification 1. Registration 2. Registration ID 6. Notification 1. Registration 2. Registration ID 5. Notification Push Service Tizen Push Server SAMSUNG TOMORROW To receive push notifications for your application: 1. Request permission to access the Tizen push servers for using the push service API. 2. Wait for a confirmation email for the request. 3. Register the installed application on the device. 4. Connect to the push service for receiving push notifications. 5. Receive notifications from the push service.

https://docs.tizen.org/application/web/guides/messaging/push/;



Push Server

You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

The main features of the Push API for the server developers include:

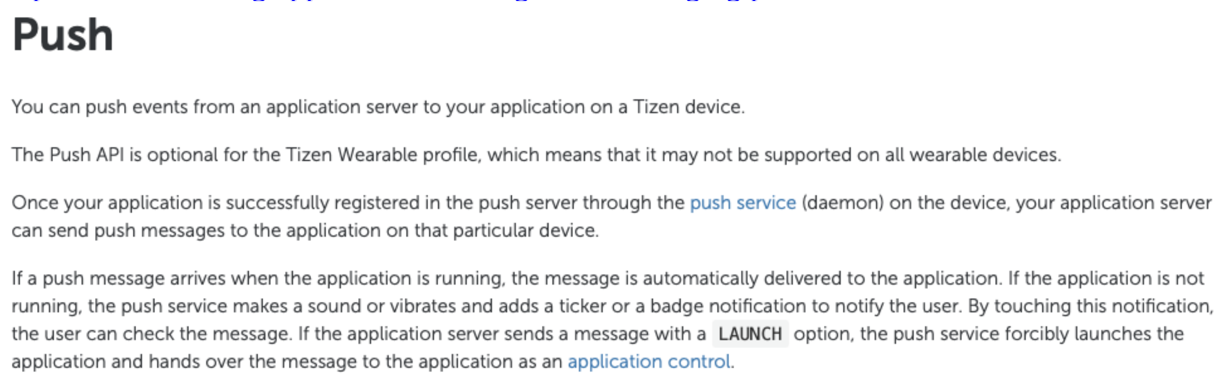
- Sending push notifications

  You can send push notifications from the application server to an application.

- Decorating push notifications

  You can add decorations to the push notifications in the quick panel.

https://docs.tizen.org/application/native/guides/messaging/push-server/;



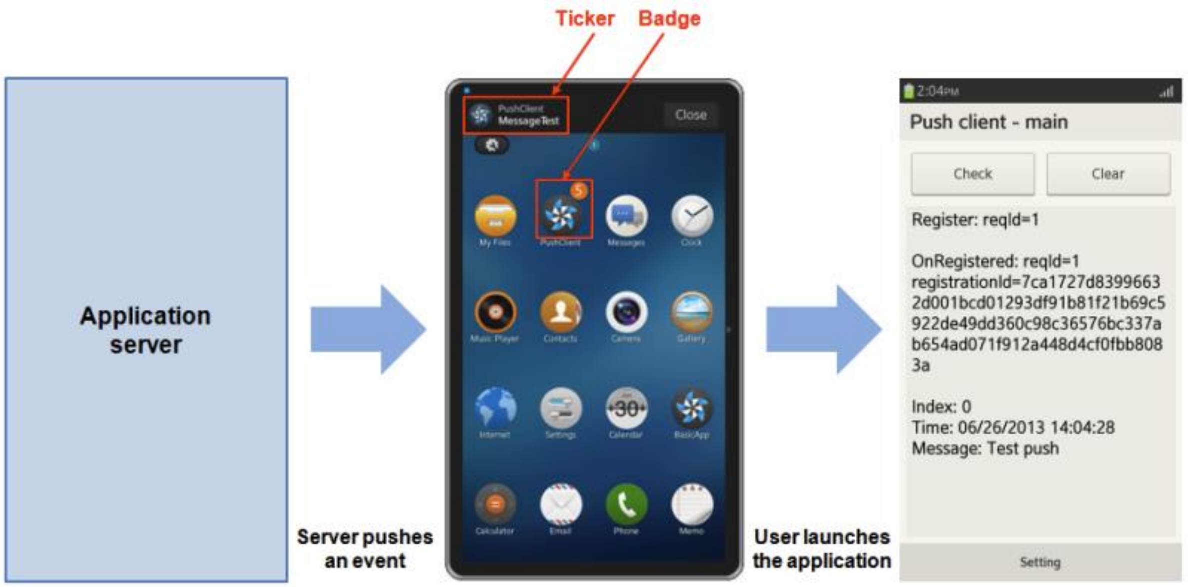
Push

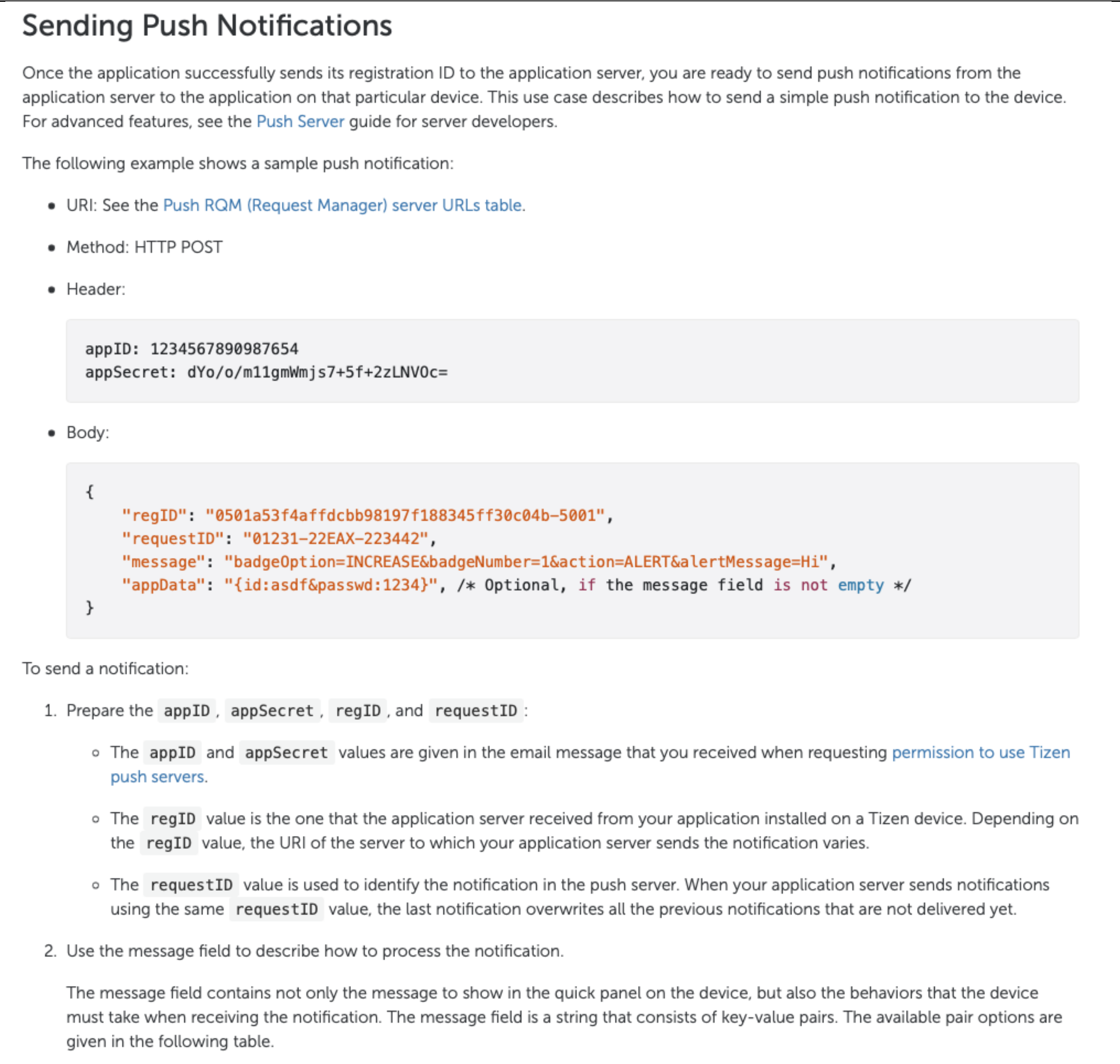
You can push events from an application server to your application on a Tizen device.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

Once your application is successfully registered in the push server through the push service (daemon) on the device, your application server can send push messages to the application on that particular device.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application as an application control.

	Figure: Push messaging service  Ticker Badge Application server Server pushes an event User launches the application PushClient MessageTest Close 2:04PM Push client - main Check Clear Setting Register: reqId=1 OnRegistered: reqId=1 registrationId=7ca1727d83996632d001bcd01293df91b81f21b69c5922de49dd360c98c36576bc337ab654ad071f912a448d4cf0fbb8083a Index: 0 Time: 06/26/2013 14:04:28 Message: Test push



Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

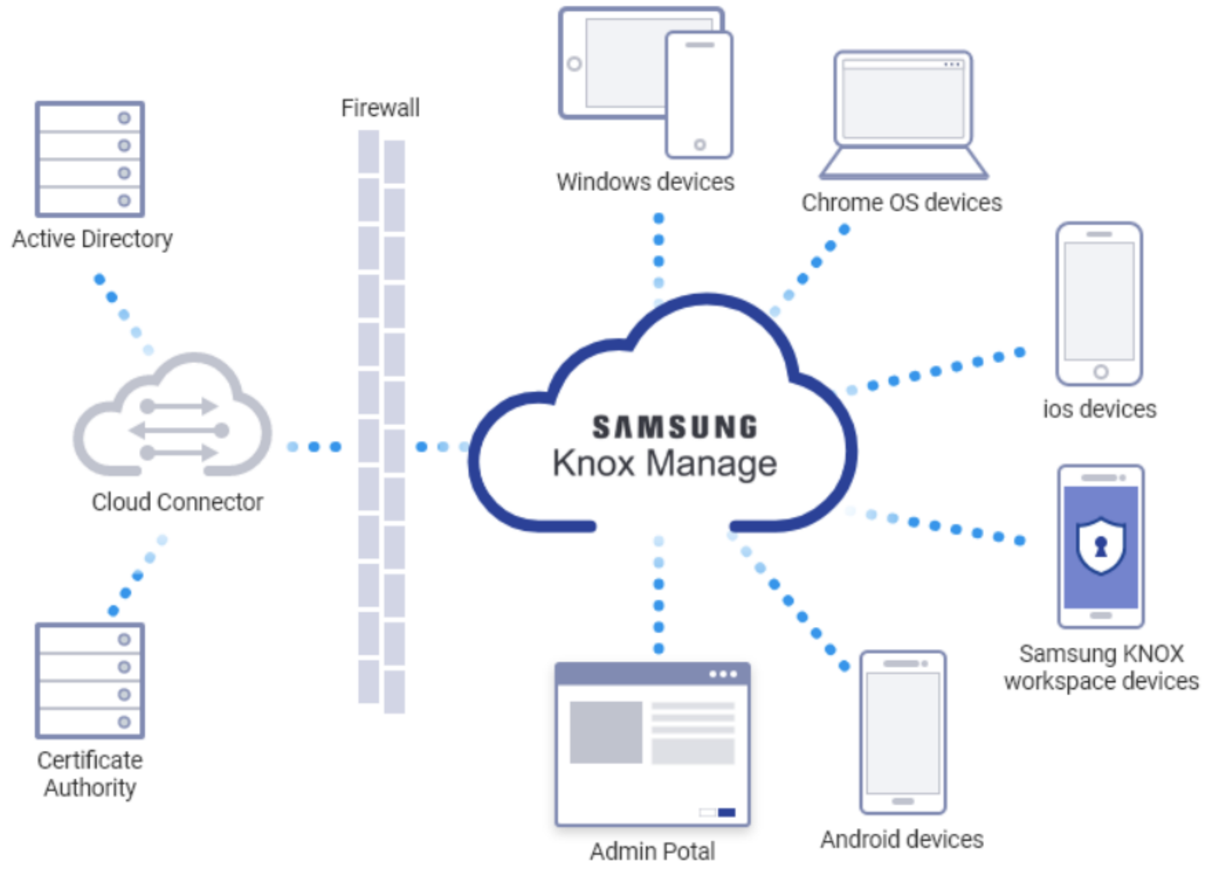
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

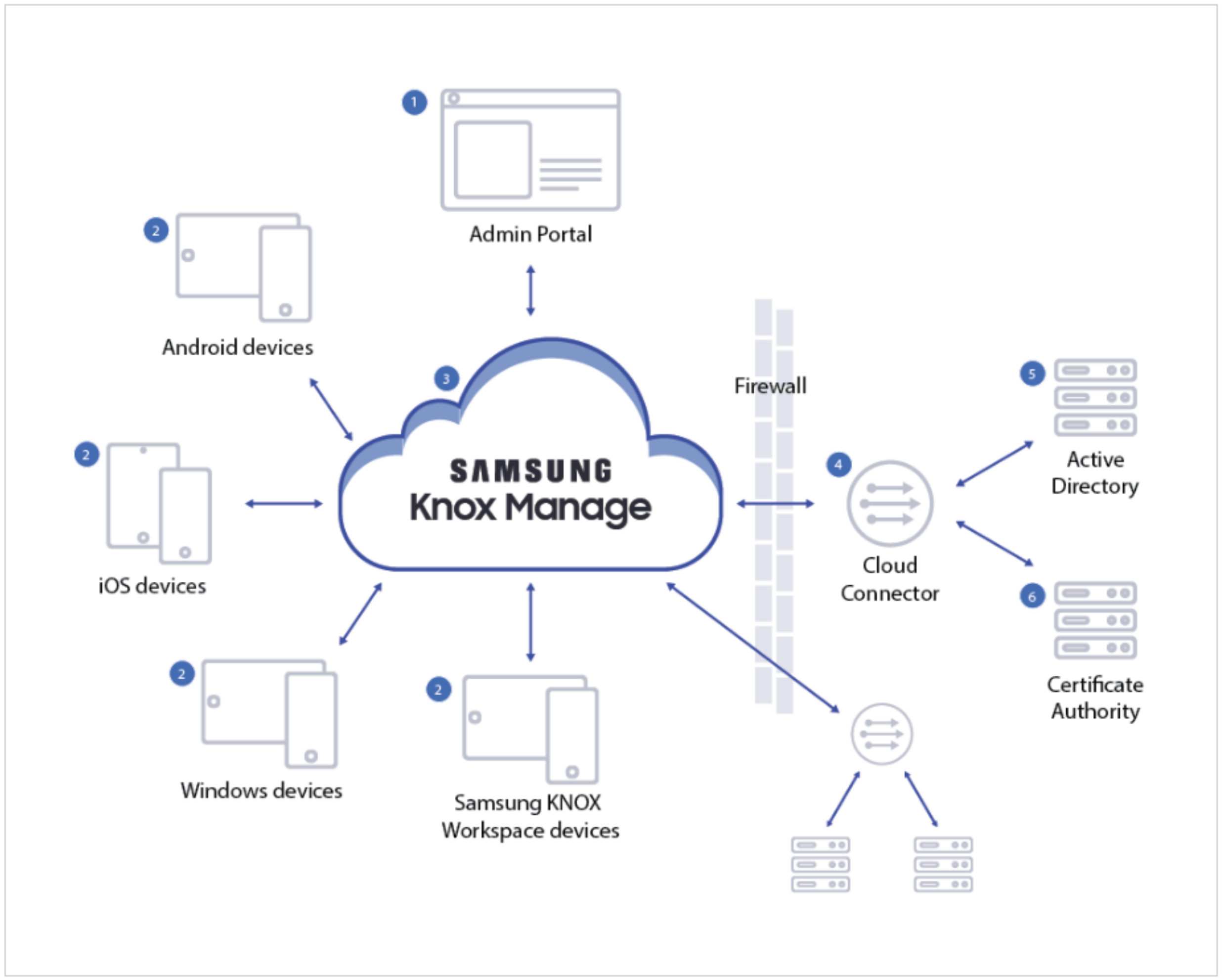
To send a notification:

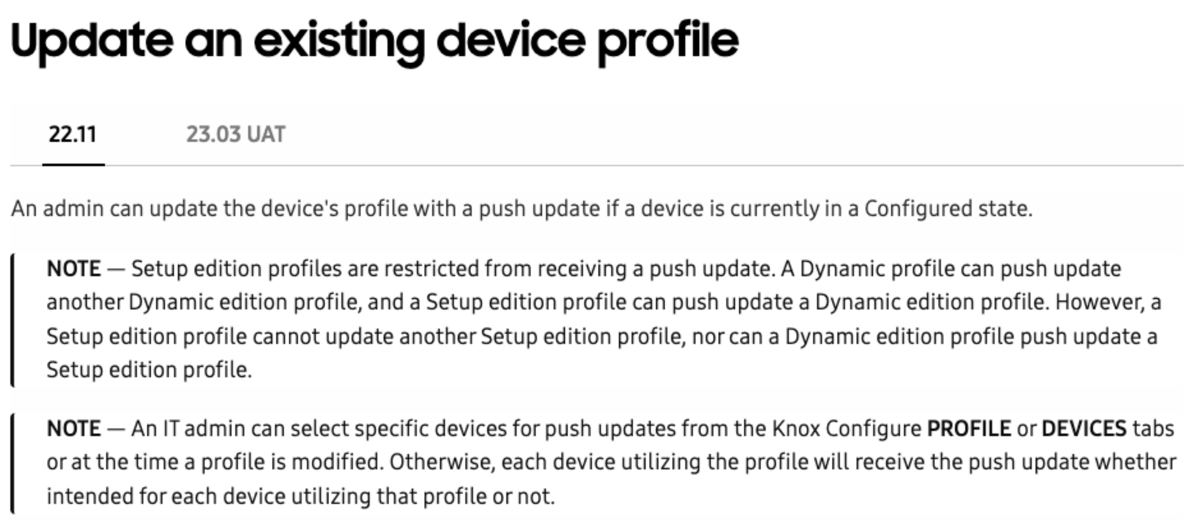
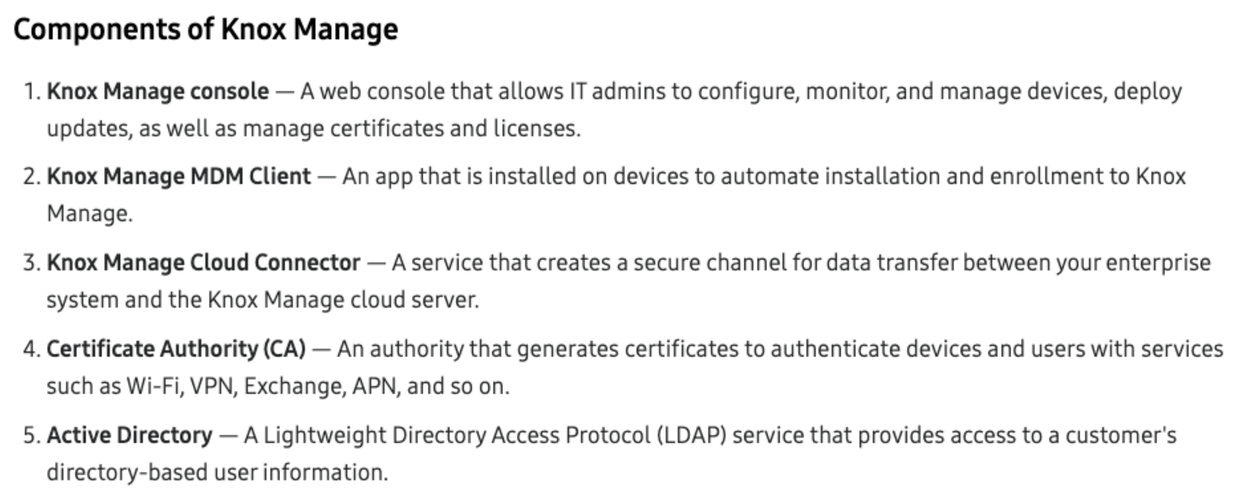
1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
   - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
   - The `regID` value is the one that the application server received from your application installed on a Tizen device. Depending on the `regID` value, the URI of the server to which your application server sends the notification varies.
   - The `requestID` value is used to identify the notification in the push server. When your application server sends notifications using the same `requestID` value, the last notification overwrites all the previous notifications that are not delivered yet.
2. Use the message field to describe how to process the notification.

   The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.

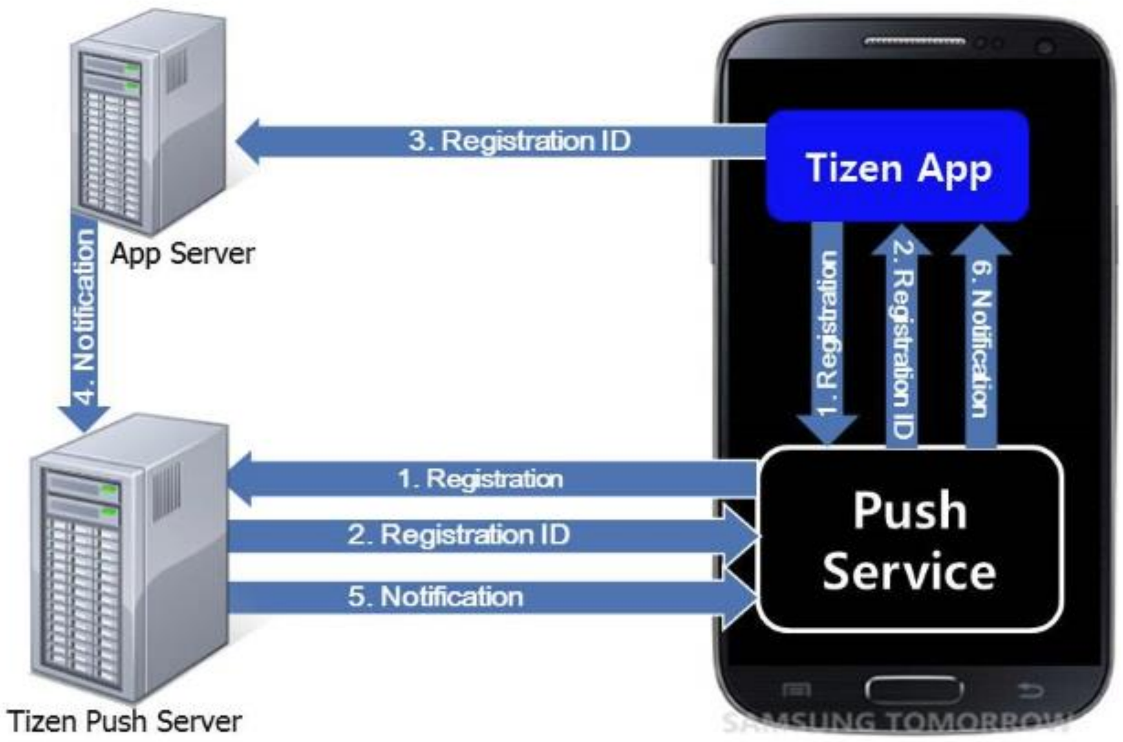
https://docs.tizen.org/application/native/guides/messaging/push/.

	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise servers which generate data messages containing an application identifier and data corresponding to the requests.  Active Directory Firewall Windows devices Chrome OS devices Cloud Connector SAMSUNG Knox Manage ios devices Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

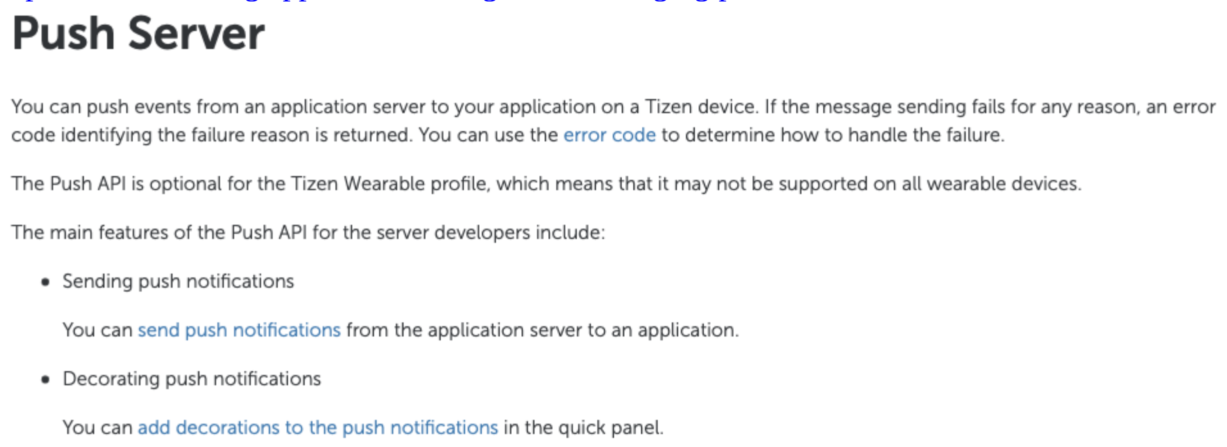
	 1 Admin Portal 2 Android devices 2 iOS devices 3 SAMSUNG Knox Manage Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority 2 Windows devices 2 Samsung KNOX Workspace devices https://docs.samsungknox.com/admin/knox-manage/km-features.htm

	 **Update an existing device profile** 22.11 23.03 UAT An admin can update the device's profile with a push update if a device is currently in a Configured state. **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile. **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not. https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm  **Components of Knox Manage** 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
[1f] the network message server to transmit each of the generated Internet data messages to the device messaging	Samsung's push messaging servers comprise "the network message server to transmit each of the generated Internet data messages to the device messaging agent located on the device

agent located on the device indicated in the corresponding request, using the corresponding secure Internet data connection for the device indicated in the corresponding request;	indicated in the corresponding request, using the corresponding secure Internet data connection for the device indicated in the corresponding request.” For example, Samsung’s push messaging servers transmit the generated data messages to the device agents on the devices. *See, e.g.*,

	## Architecture The architecture of the Tizen Push service is described in detail in the mobile native Push guide. Figure: Service architecture  3. Registration ID Tizen App App Server 4. Notification 1. Registration 2. Registration ID 6. Notification 1. Registration 2. Registration ID 5. Notification Push Service Tizen Push Server SAMSUNG TOMORROW To receive push notifications for your application: 1. Request permission to access the Tizen push servers for using the push service API. 2. Wait for a confirmation email for the request. 3. Register the installed application on the device. 4. Connect to the push service for receiving push notifications. 5. Receive notifications from the push service.

https://docs.tizen.org/application/web/guides/messaging/push/;



Push Server

You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

The main features of the Push API for the server developers include:

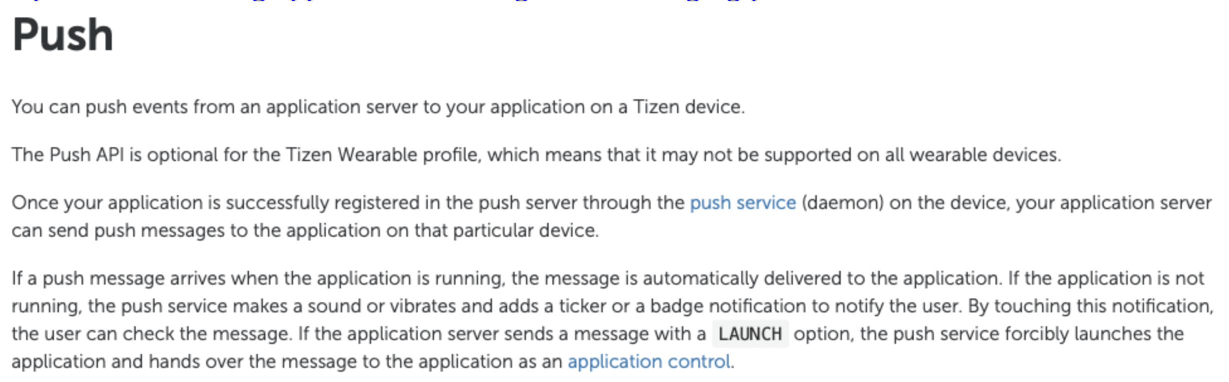
- Sending push notifications

  You can send push notifications from the application server to an application.

- Decorating push notifications

  You can add decorations to the push notifications in the quick panel.

https://docs.tizen.org/application/native/guides/messaging/push-server/;



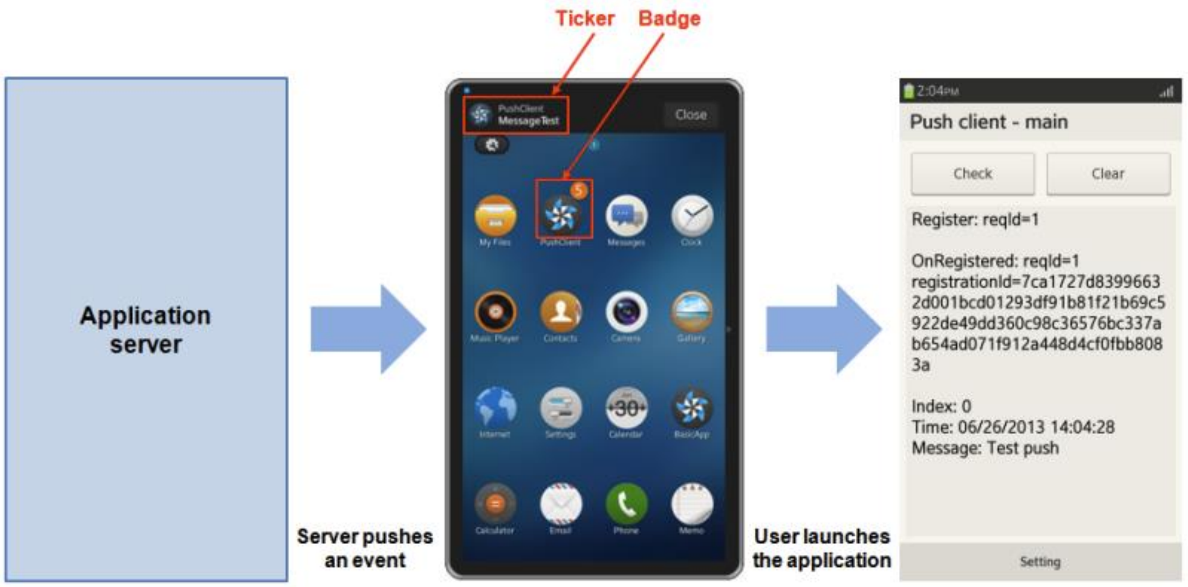
Push

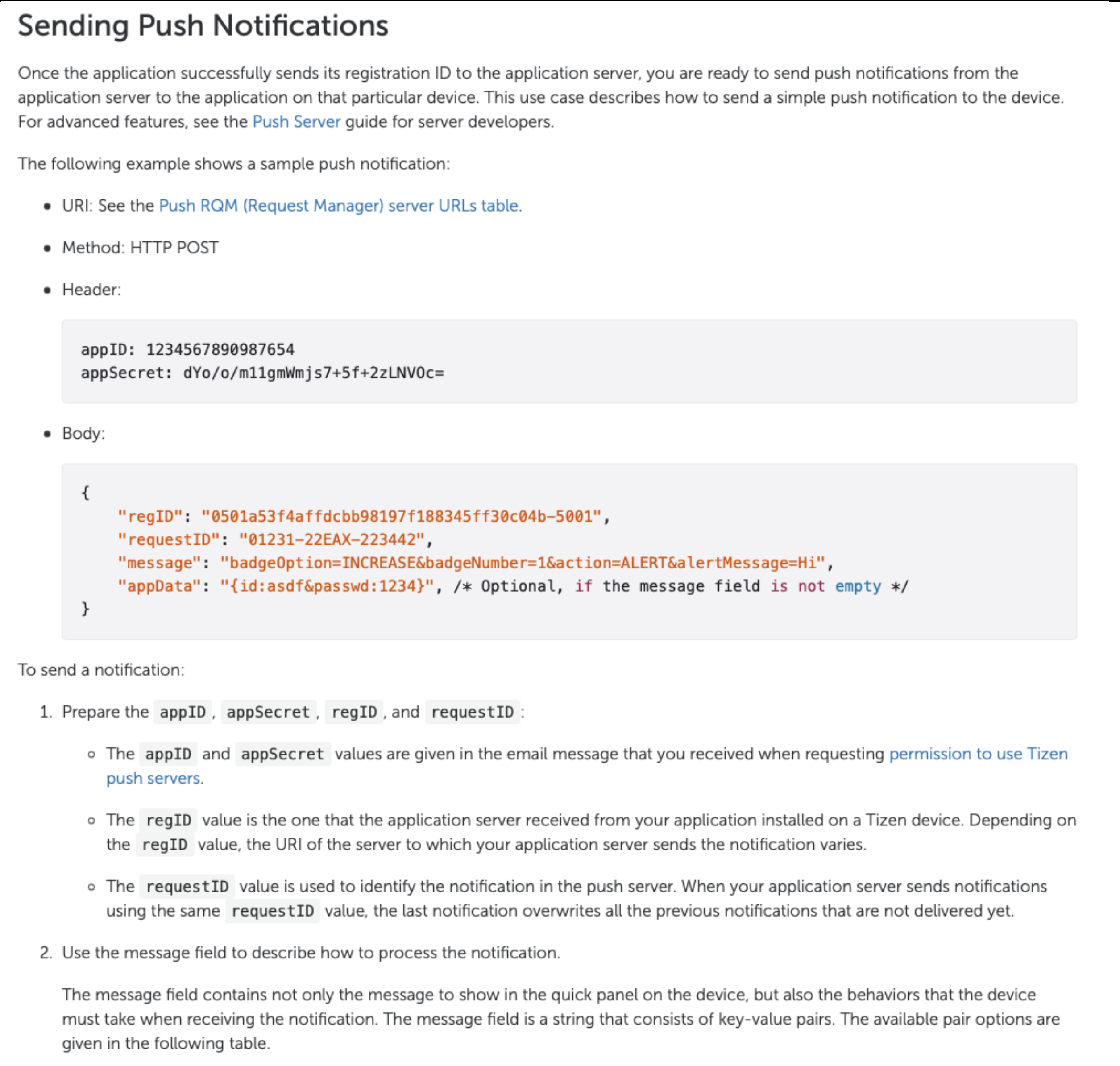
You can push events from an application server to your application on a Tizen device.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

Once your application is successfully registered in the push server through the push service (daemon) on the device, your application server can send push messages to the application on that particular device.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application as an application control.

	Figure: Push messaging service  Ticker Badge Application server Server pushes an event User launches the application PushClient MessageTest Close My Files PushClient Messages Clock Music Player Contacts Camera Gallery Internet Settings Calendar BasicApp Calculator Email Phone Memo 2:04PM Push client - main Check Clear Setting Register: reqId=1 OnRegistered: reqId=1 registrationId=7ca1727d8399663 2d001bcd01293df91b81f21b69c5 922de49dd360c98c36576bc337a b654ad071f912a448d4cf0fbb808 3a Index: 0 Time: 06/26/2013 14:04:28 Message: Test push



Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

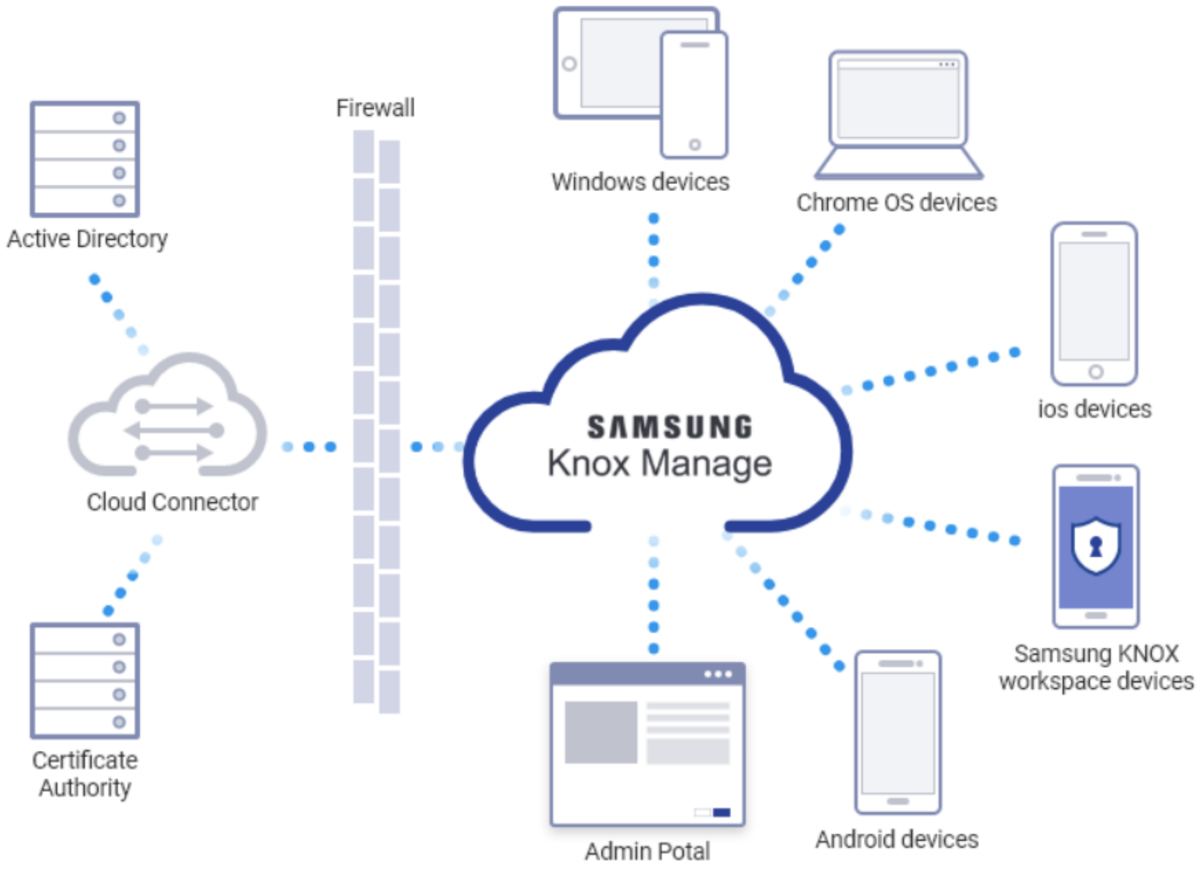
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

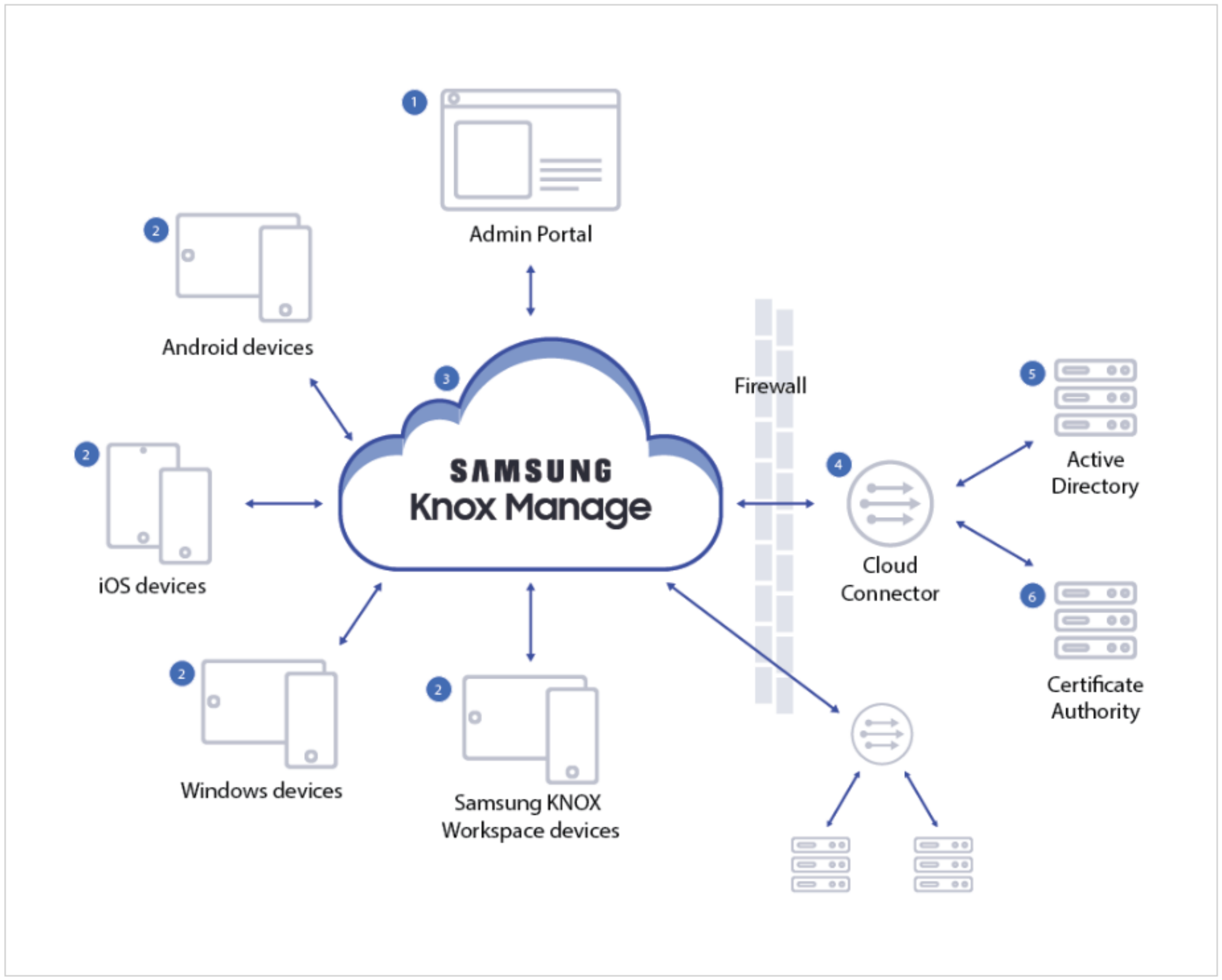
To send a notification:

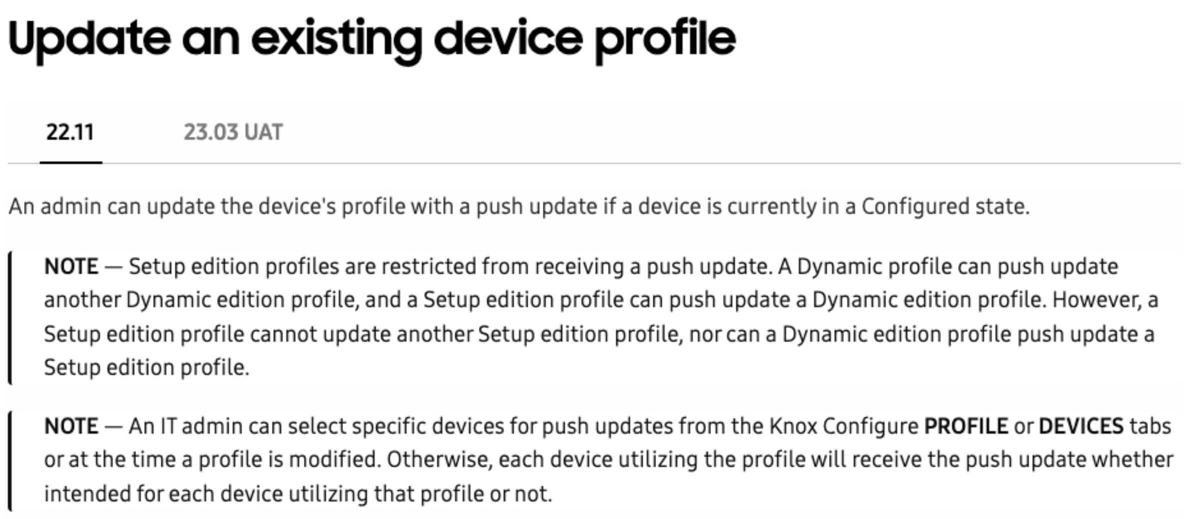
1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
   - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
   - The `regID` value is the one that the application server received from your application installed on a Tizen device. Depending on the `regID` value, the URI of the server to which your application server sends the notification varies.
   - The `requestID` value is used to identify the notification in the push server. When your application server sends notifications using the same `requestID` value, the last notification overwrites all the previous notifications that are not delivered yet.
2. Use the message field to describe how to process the notification.

   The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.

https://docs.tizen.org/application/native/guides/messaging/push/.

	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise servers which transmit the generated data messages to the device agents on the devices.  Firewall Active Directory Cloud Connector Certificate Authority Windows devices Chrome OS devices SAMSUNG Knox Manage ios devices Samsung KNOX workspace devices Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 1 Admin Portal 2 Android devices 2 iOS devices 3 SAMSUNG Knox Manage Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority 2 Windows devices 2 Samsung KNOX Workspace devices https://docs.samsungknox.com/admin/knox-manage/km-features.htm



Update an existing device profile

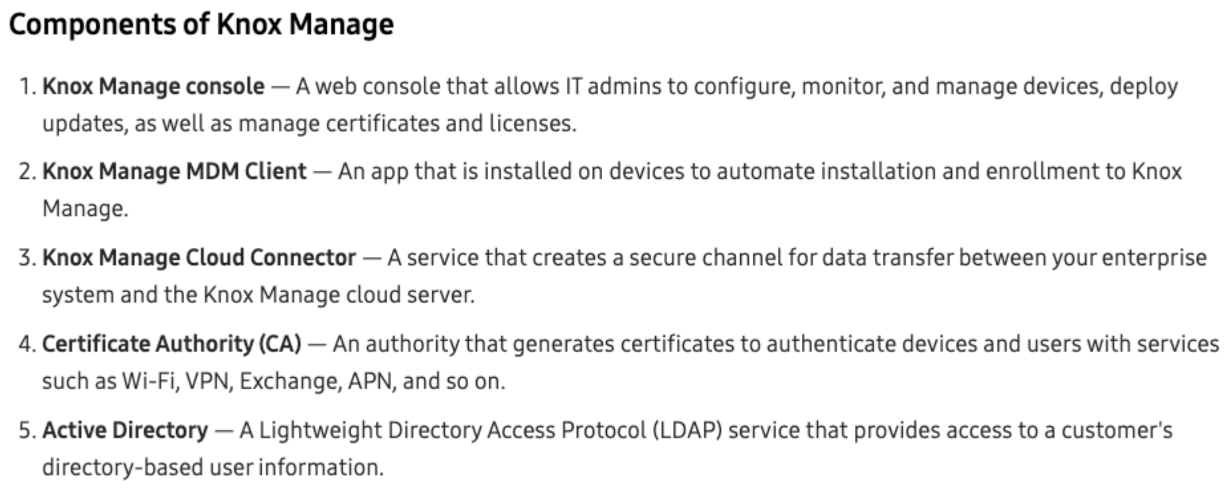
22.11 23.03 UAT

An admin can update the device's profile with a push update if a device is currently in a Configured state.

> **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

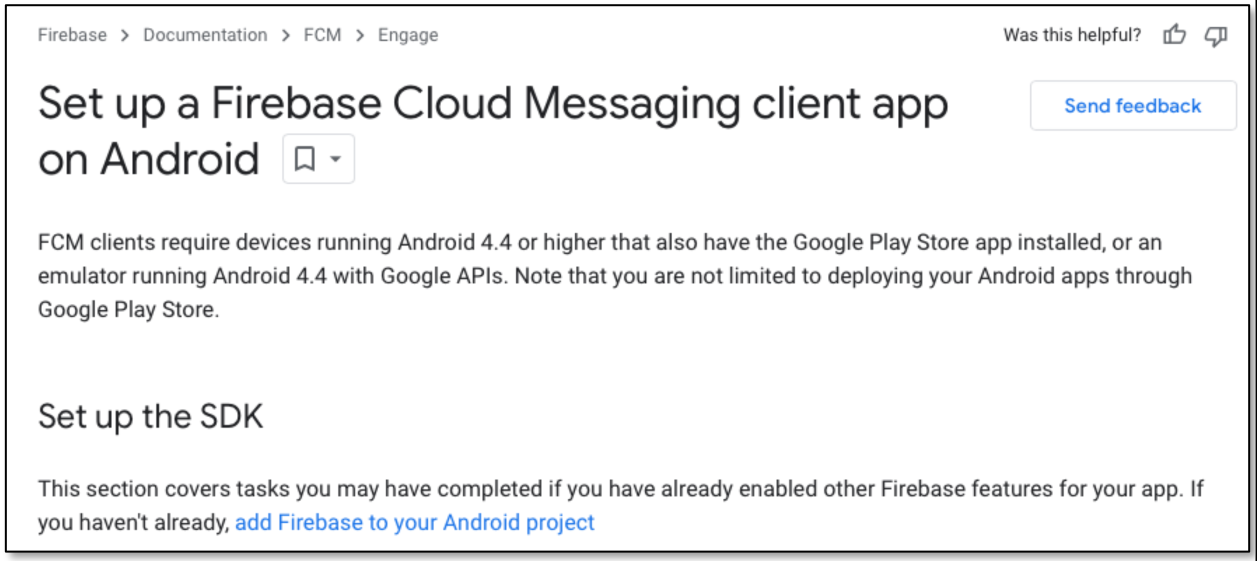
https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm

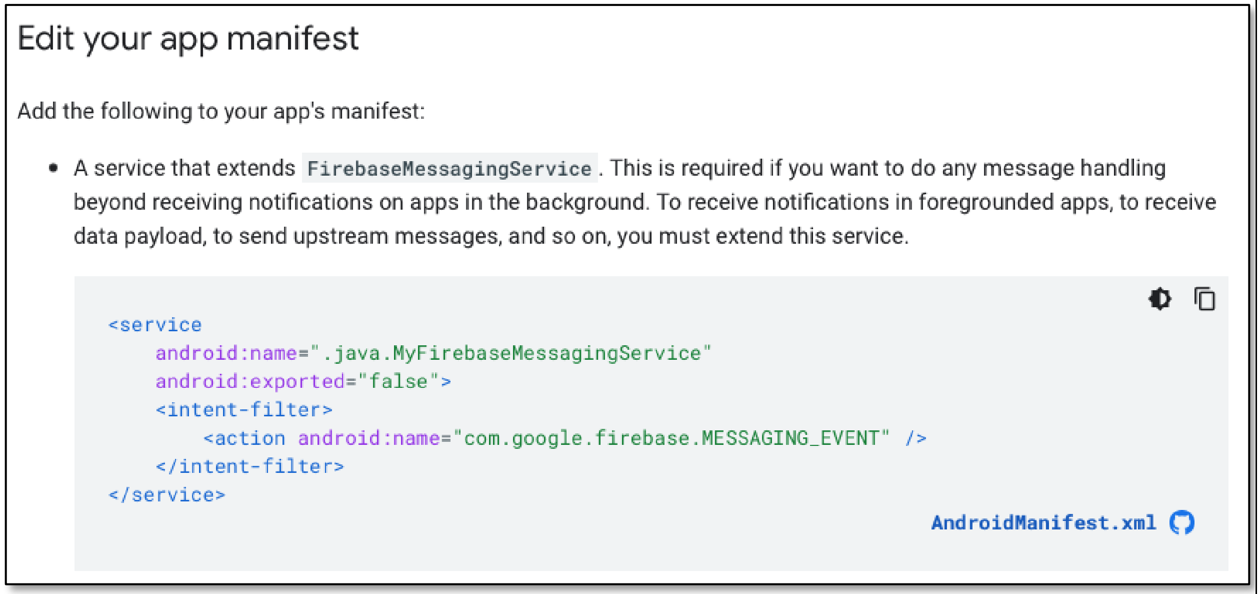


Components of Knox Manage

1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses.
2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage.
3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server.
4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on.
5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information.

https://docs.samsungknox.com/admin/knox-manage/welcome.htm

[1g] each device messaging agent, when executing, to receive the Internet data messages from the secure Internet data connection corresponding to the device executing the device messaging agent, and	Samsung's devices each comprise a "device messaging agent, when executing, to receive the Internet data messages from the secure Internet data connection corresponding to the device executing the device messaging agent." For example, Samsung devices receive data over a secure internet data connection. *See e.g.*,  Firebase > Documentation > FCM > Engage Set up a Firebase Cloud Messaging client app on Android FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store. Set up the SDK This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project



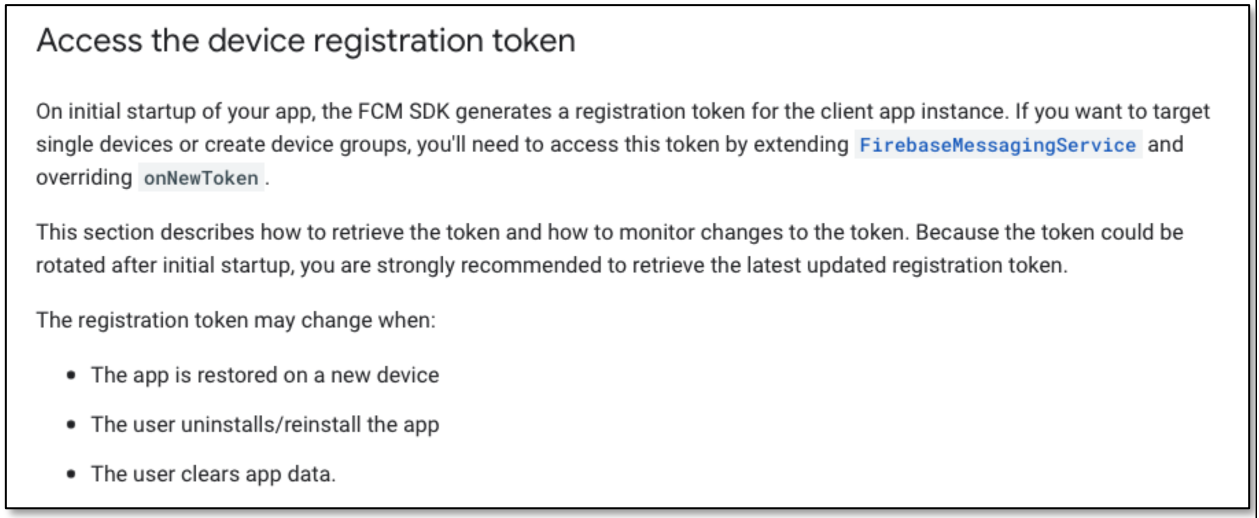
Edit your app manifest

Add the following to your app's manifest:

- A service that extends `FirebaseMessagingService`. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service.

```
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

AndroidManifest.xml



Access the device registration token

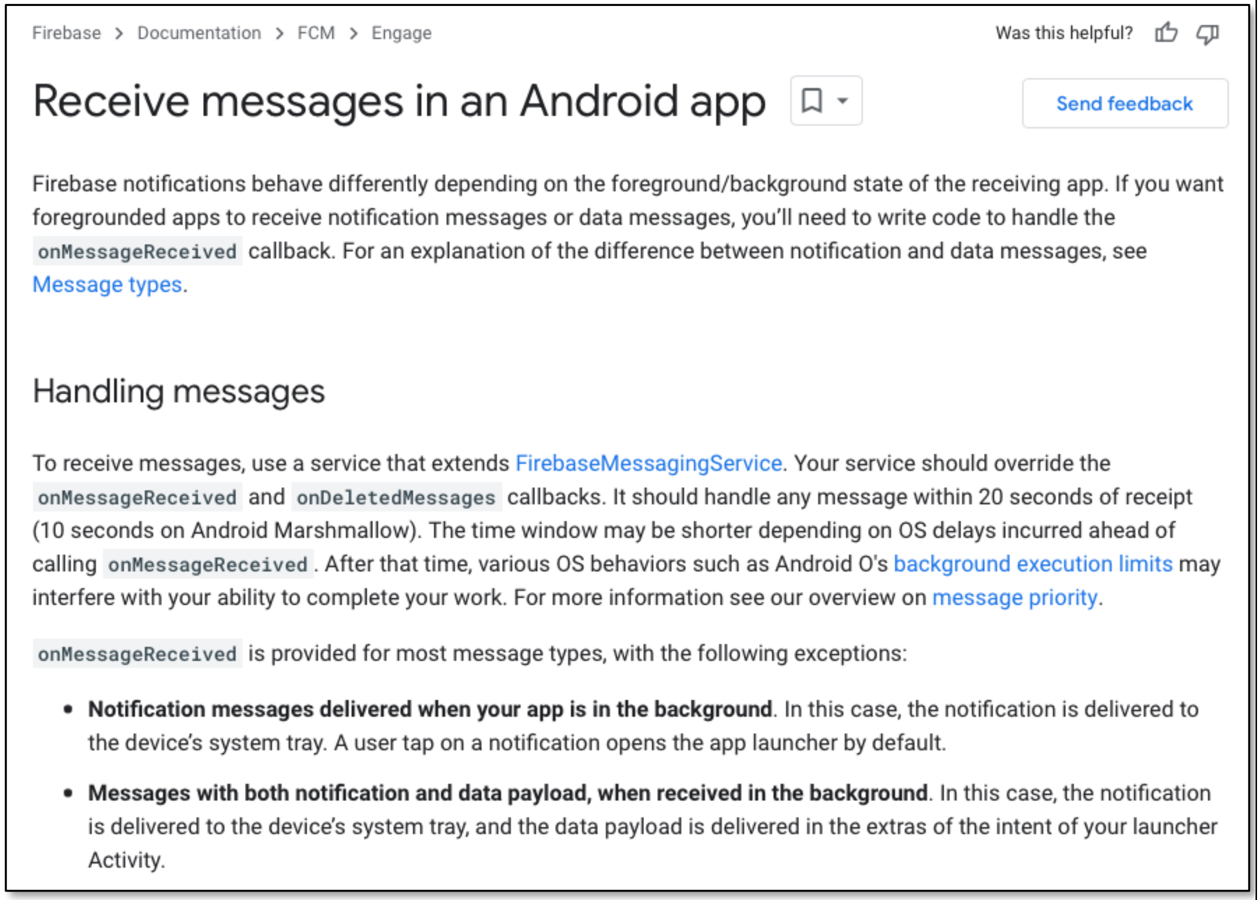
On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`.

This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

https://firebase.google.com/docs/cloud-messaging/android/client;



Receive messages in an Android app

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see Message types.

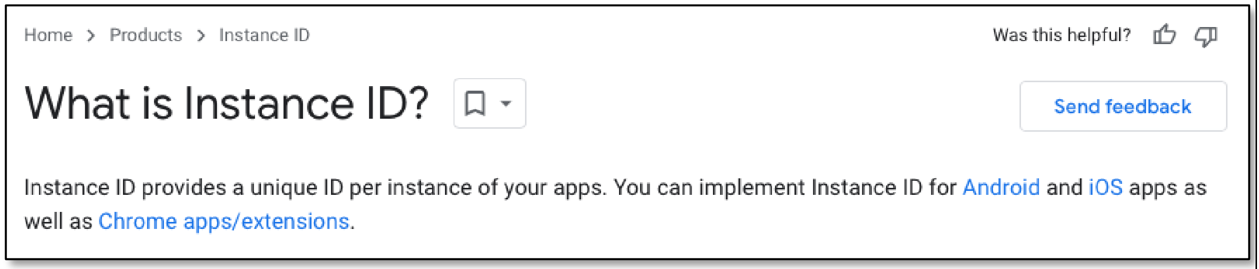
Handling messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's background execution limits may interfere with your ability to complete your work. For more information see our overview on message priority.

`onMessageReceived` is provided for most message types, with the following exceptions:

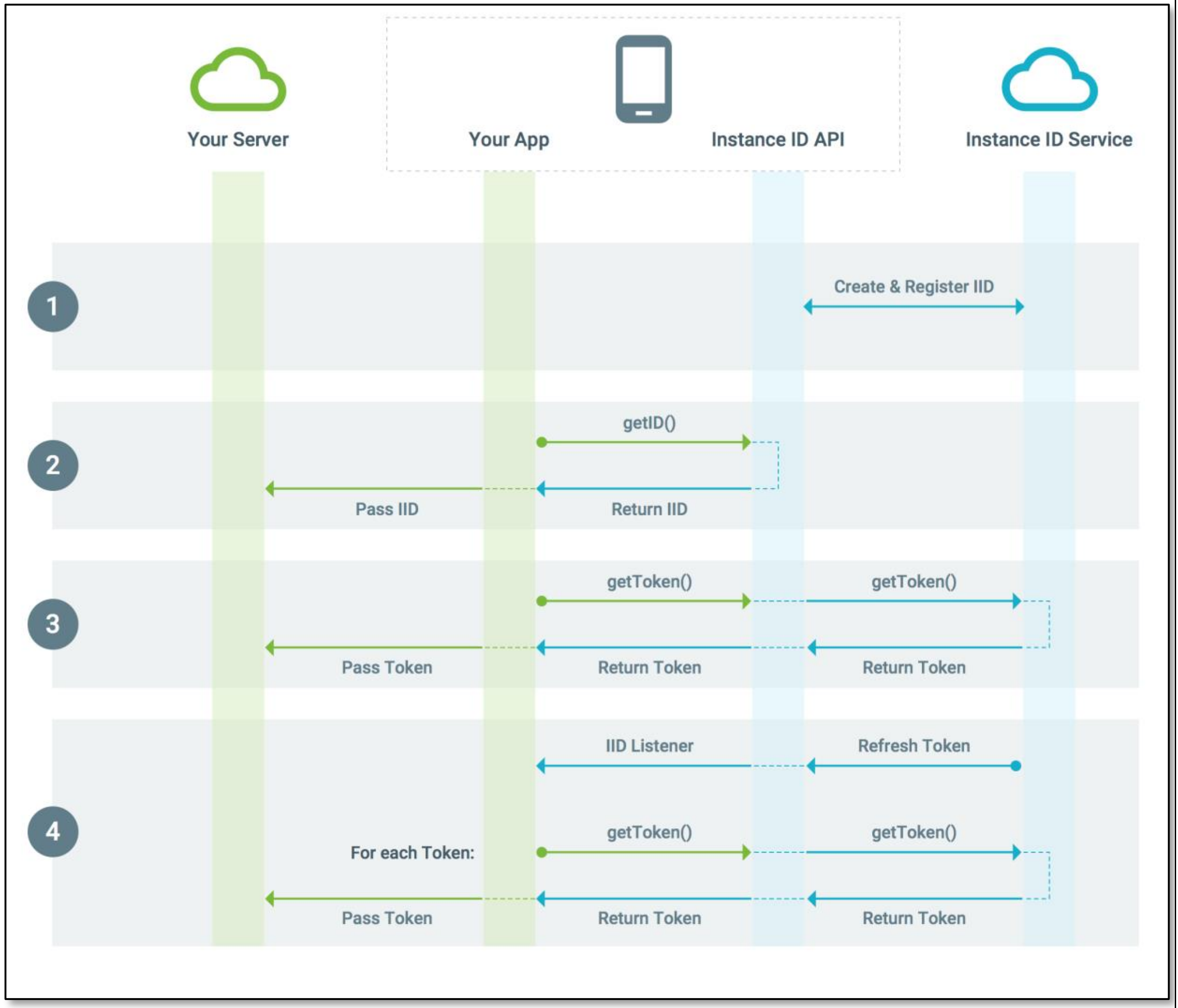
- **Notification messages delivered when your app is in the background**. In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

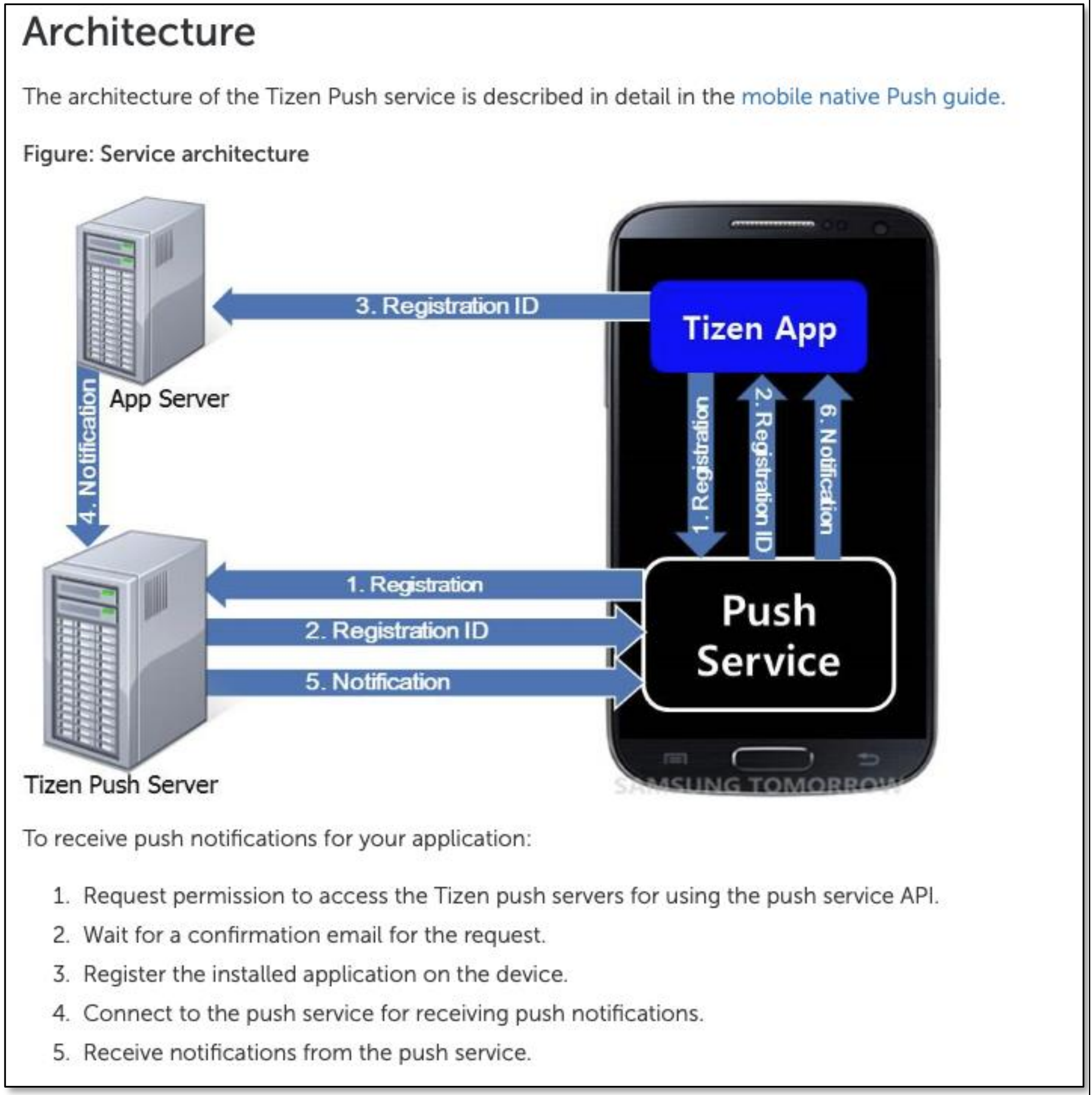
https://firebase.google.com/docs/cloud-messaging/android/receive;



What is Instance ID?

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for Android and iOS apps as well as Chrome apps/extensions.

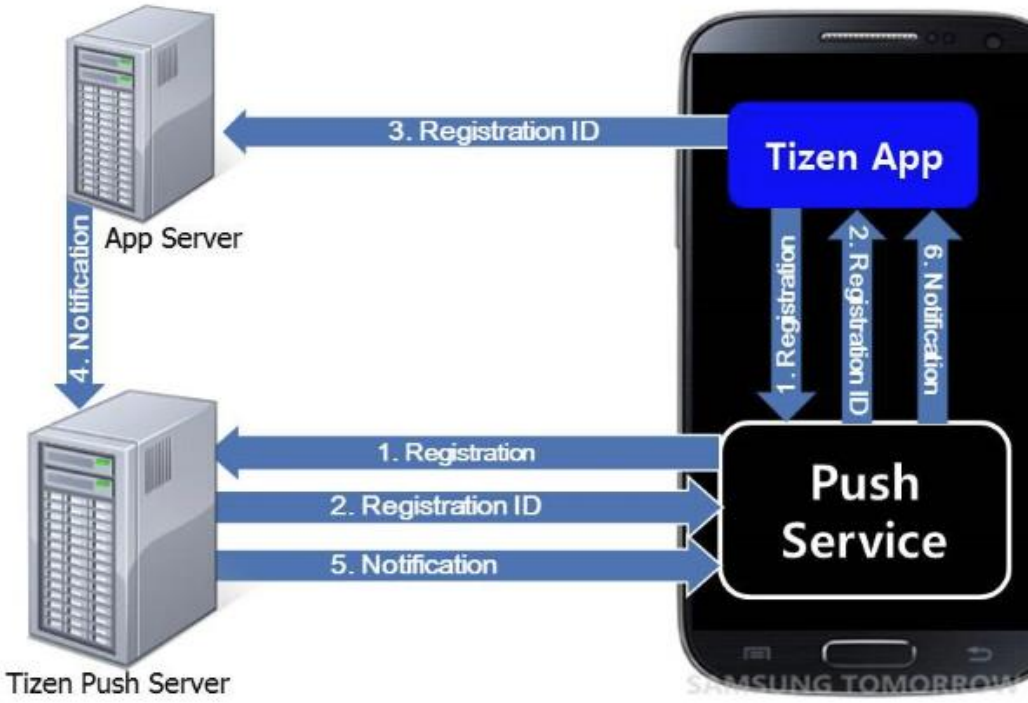
	 Your Server Your App Instance ID API Instance ID Service 1 Create & Register IID 2 getID() Pass IID Return IID 3 getToken() getToken() Pass Token Return Token Return Token 4 IID Listener Refresh Token For each Token: getToken() getToken() Pass Token Return Token Return Token https://developers.google.com/instance-id. For further example, Samsung's Tizen based devices receive data over a secure internet data connection. *See e.g.*,



Architecture

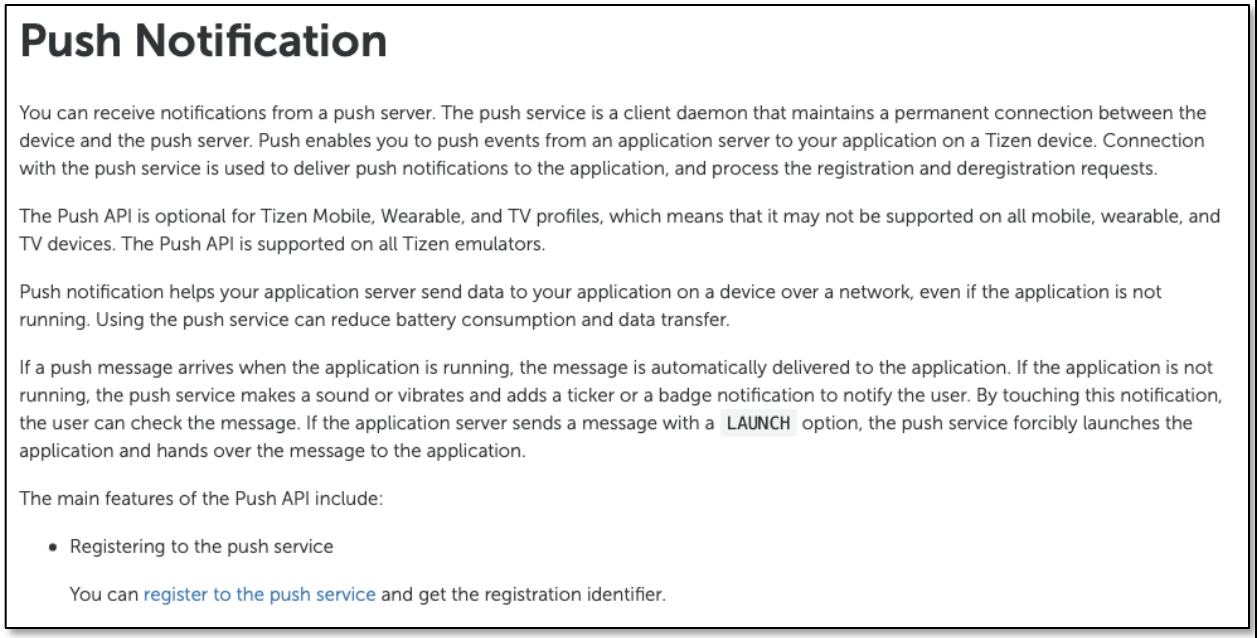
The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.



Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

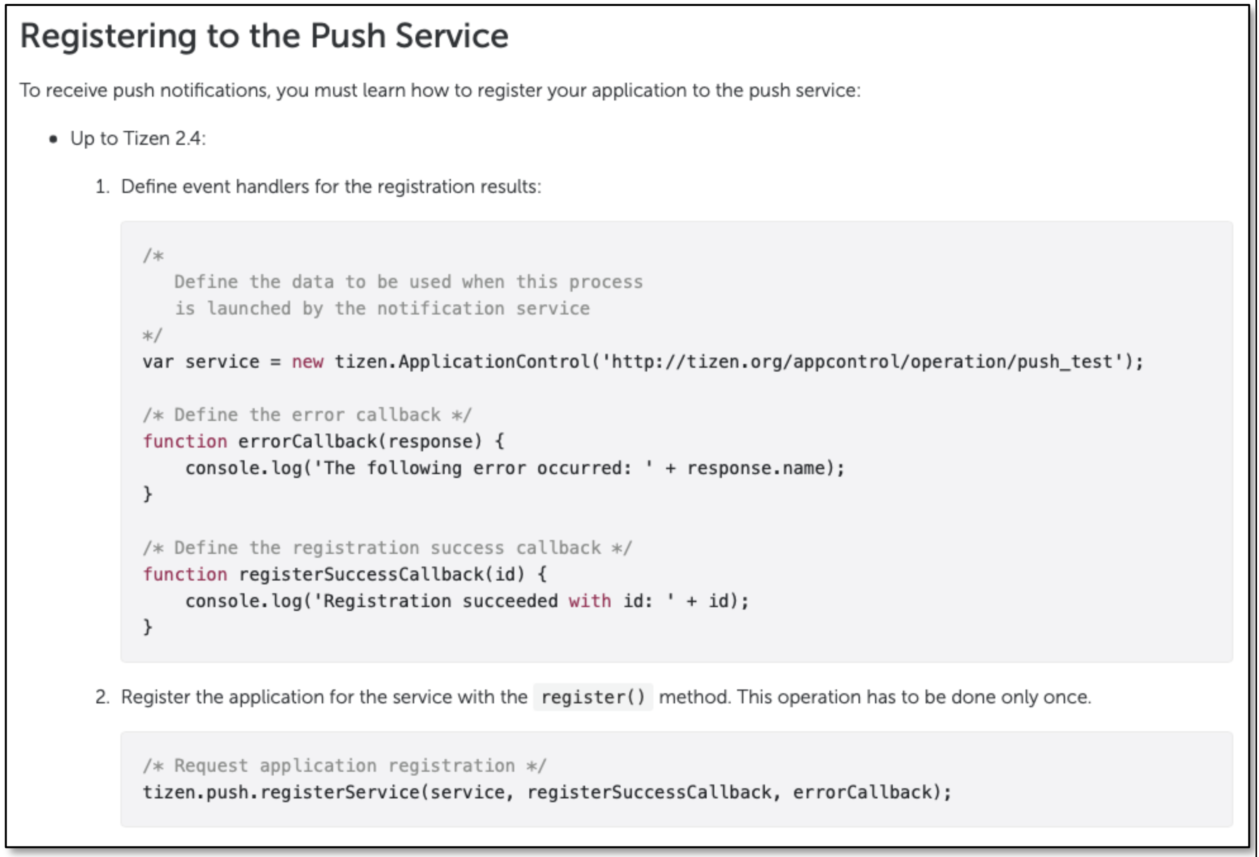
Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

  You can register to the push service and get the registration identifier.



Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

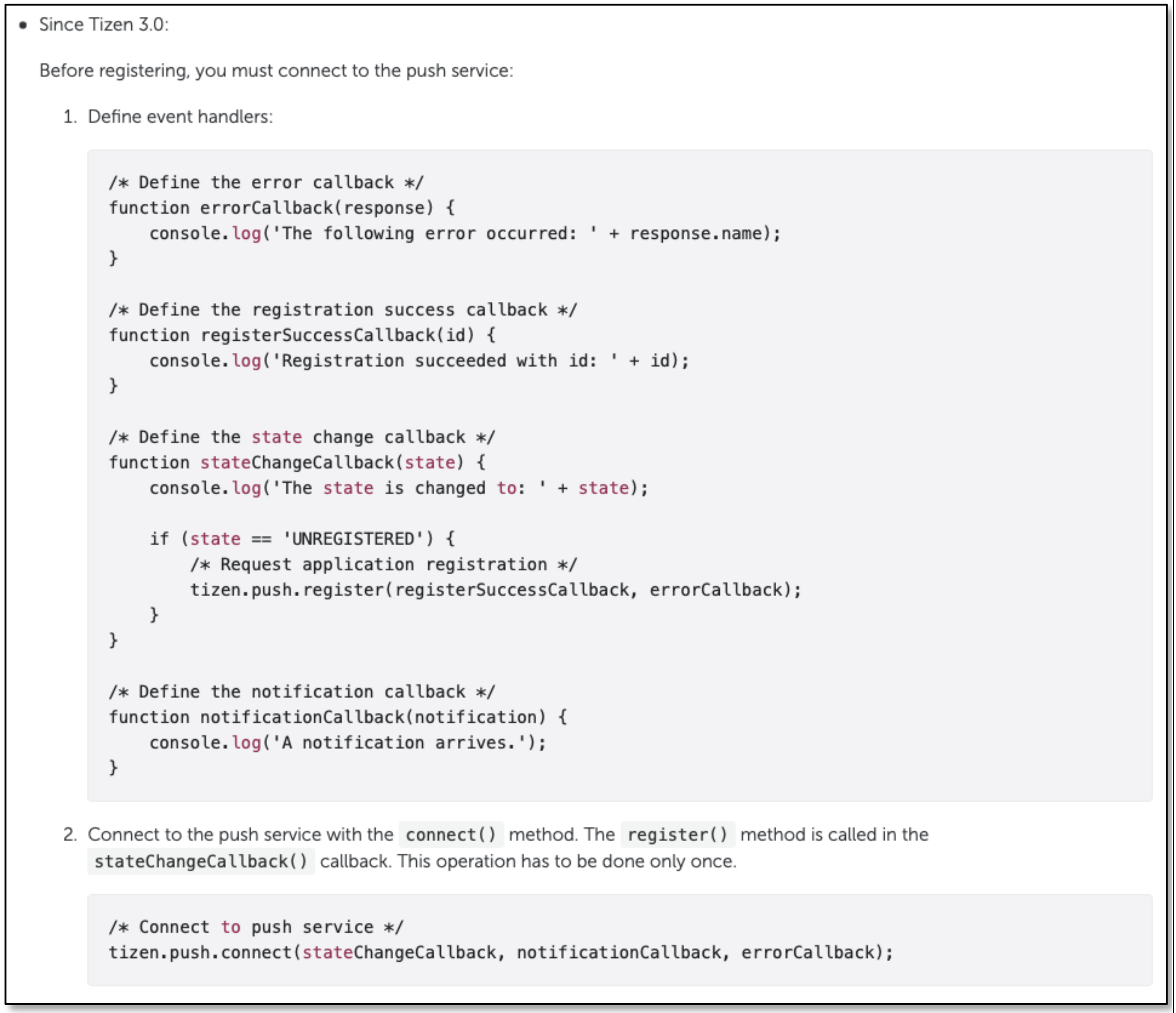
- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

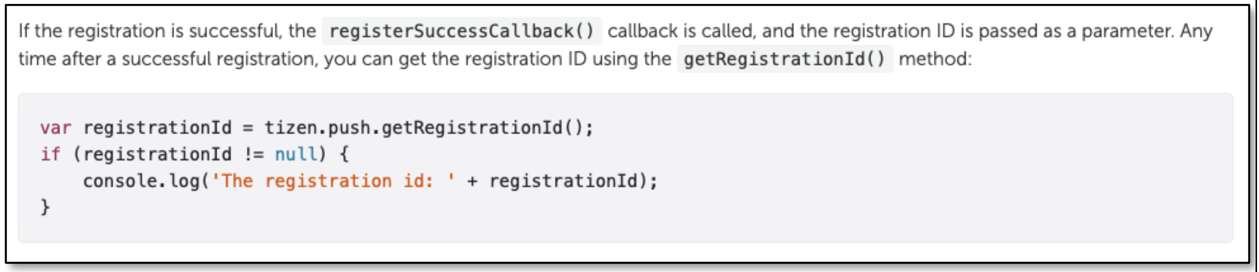
  Before registering, you must connect to the push service:

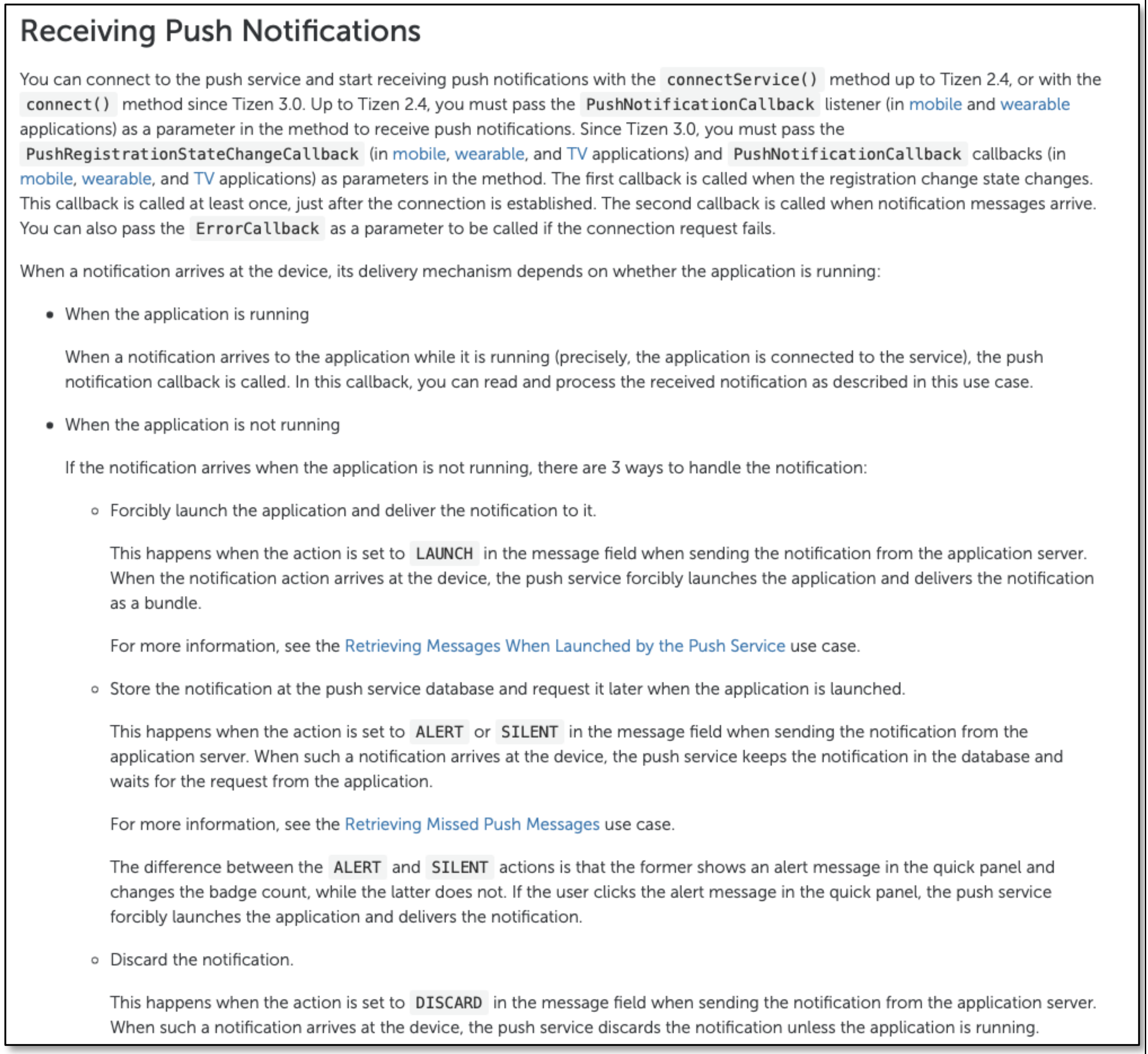
  1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

	 If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:
	`var registrationId = tizen.push.getRegistrationId();`
	`if (registrationId != null) {`
	`    console.log('The registration id: ' + registrationId);`
	`}`



Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in mobile and wearable applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the `PushRegistrationStateChangeCallback` (in mobile, wearable, and TV applications) and `PushNotificationCallback` callbacks (in mobile, wearable, and TV applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

  When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

  If the notification arrives when the application is not running, there are 3 ways to handle the notification:

  - Forcibly launch the application and deliver the notification to it.

    This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

  - Store the notification at the push service database and request it later when the application is launched.

    This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

    For more information, see the Retrieving Missed Push Messages use case.

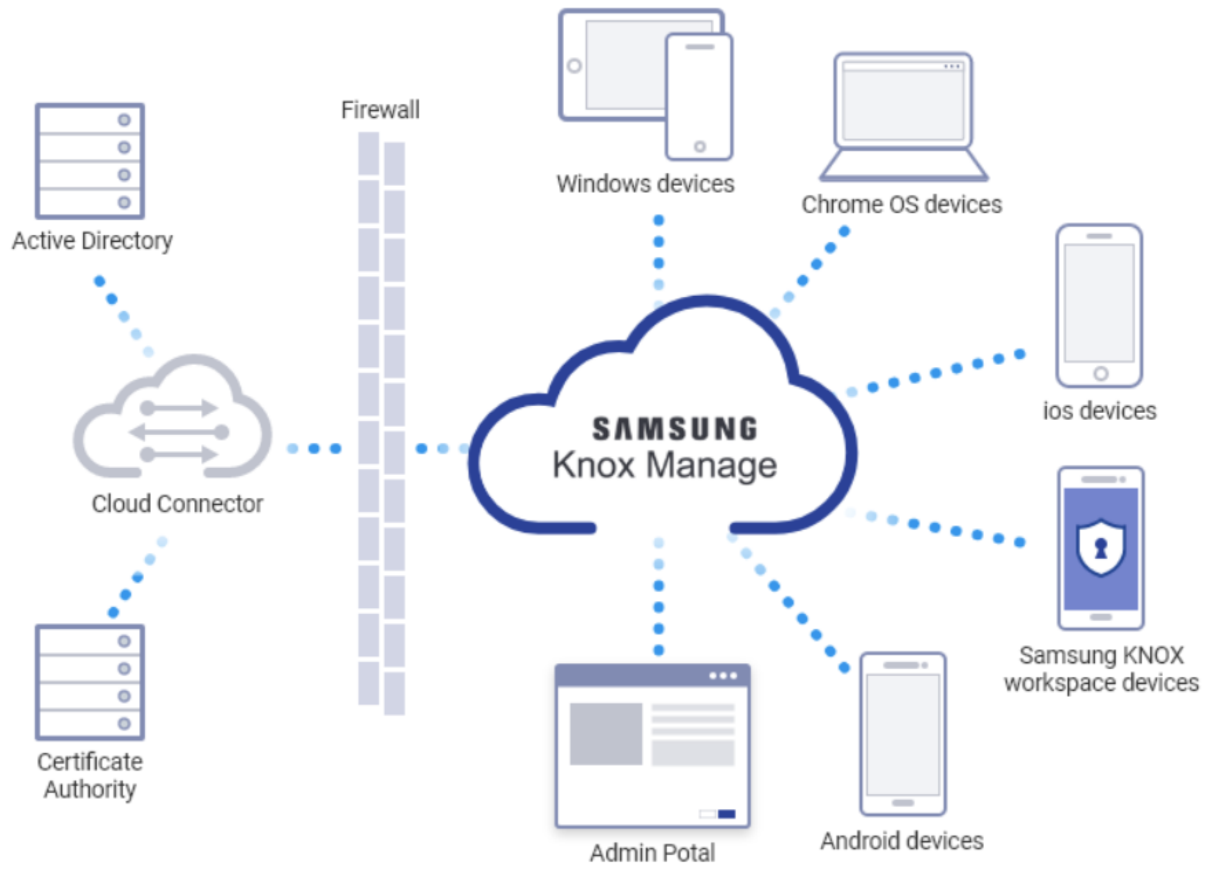
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

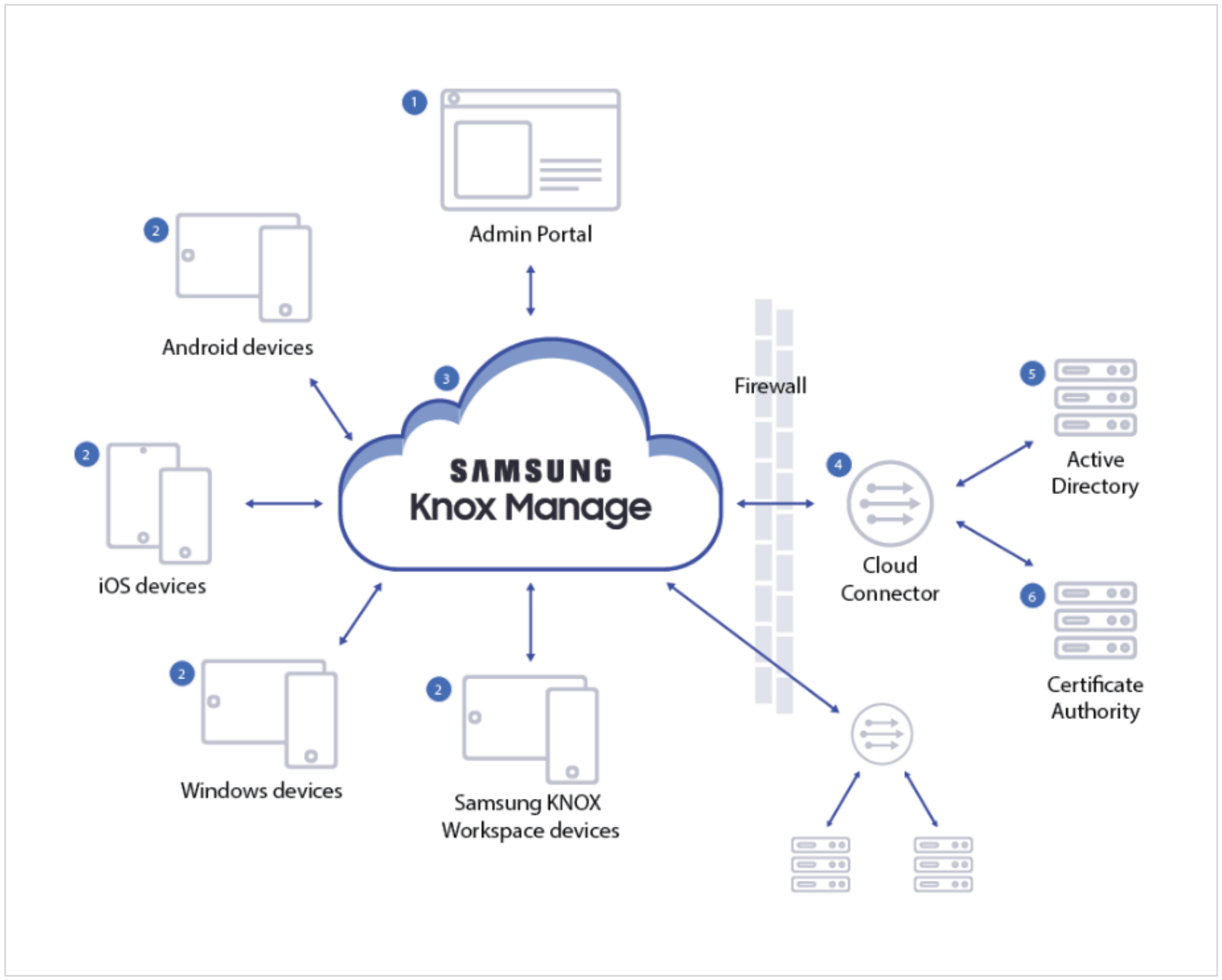
  - Discard the notification.

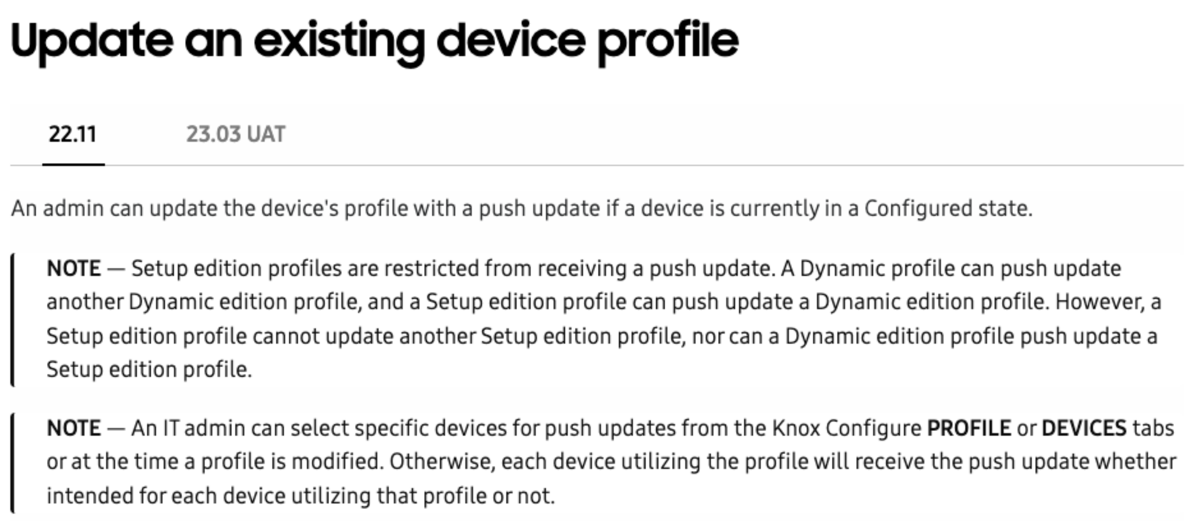
    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

https://docs.tizen.org/application/web/guides/messaging/push/.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the

	Samsung Knox servers managing those phones and devices, comprise devices receiving data over a secure connection.  Firewall Windows devices Chrome OS devices Active Directory ios devices SAMSUNG Knox Manage Cloud Connector Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 1 Admin Portal 2 Android devices 2 iOS devices 3 SAMSUNG Knox Manage Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority 2 Windows devices 2 Samsung KNOX Workspace devices https://docs.samsungknox.com/admin/knox-manage/km-features.htm



Update an existing device profile

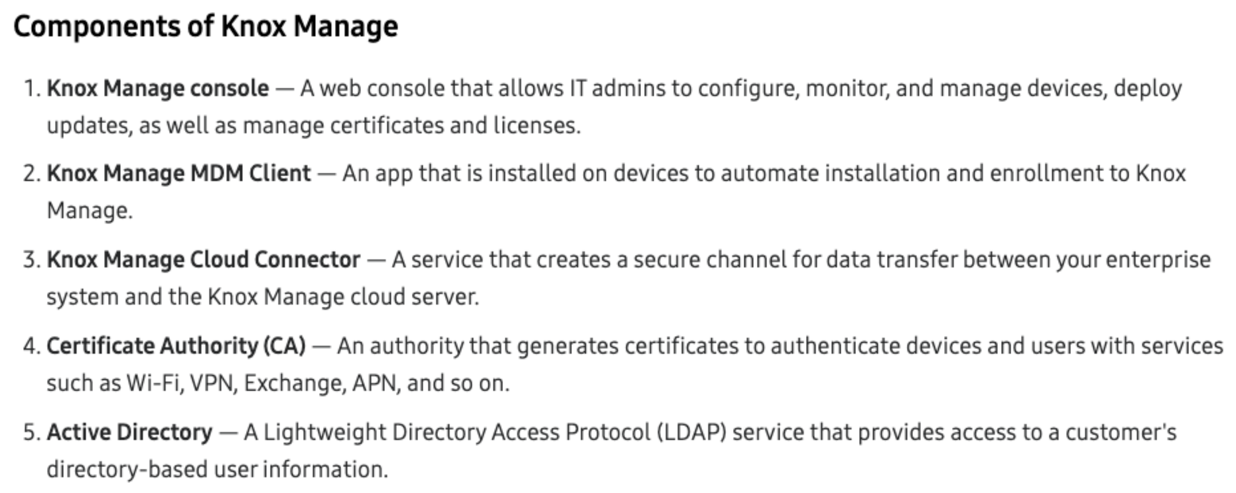
22.11 23.03 UAT

An admin can update the device's profile with a push update if a device is currently in a Configured state.

> **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

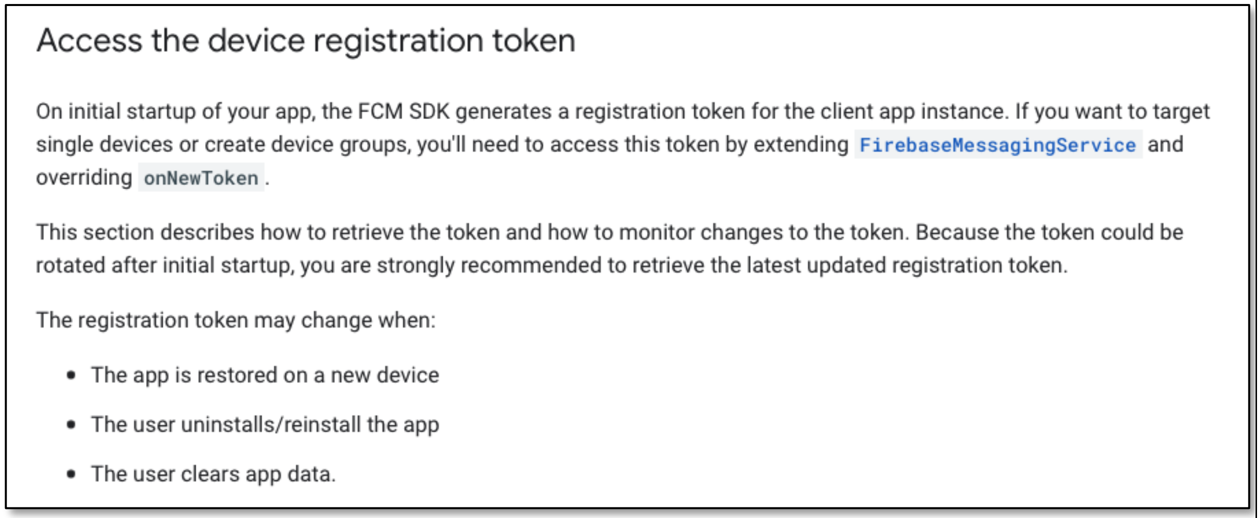
https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm

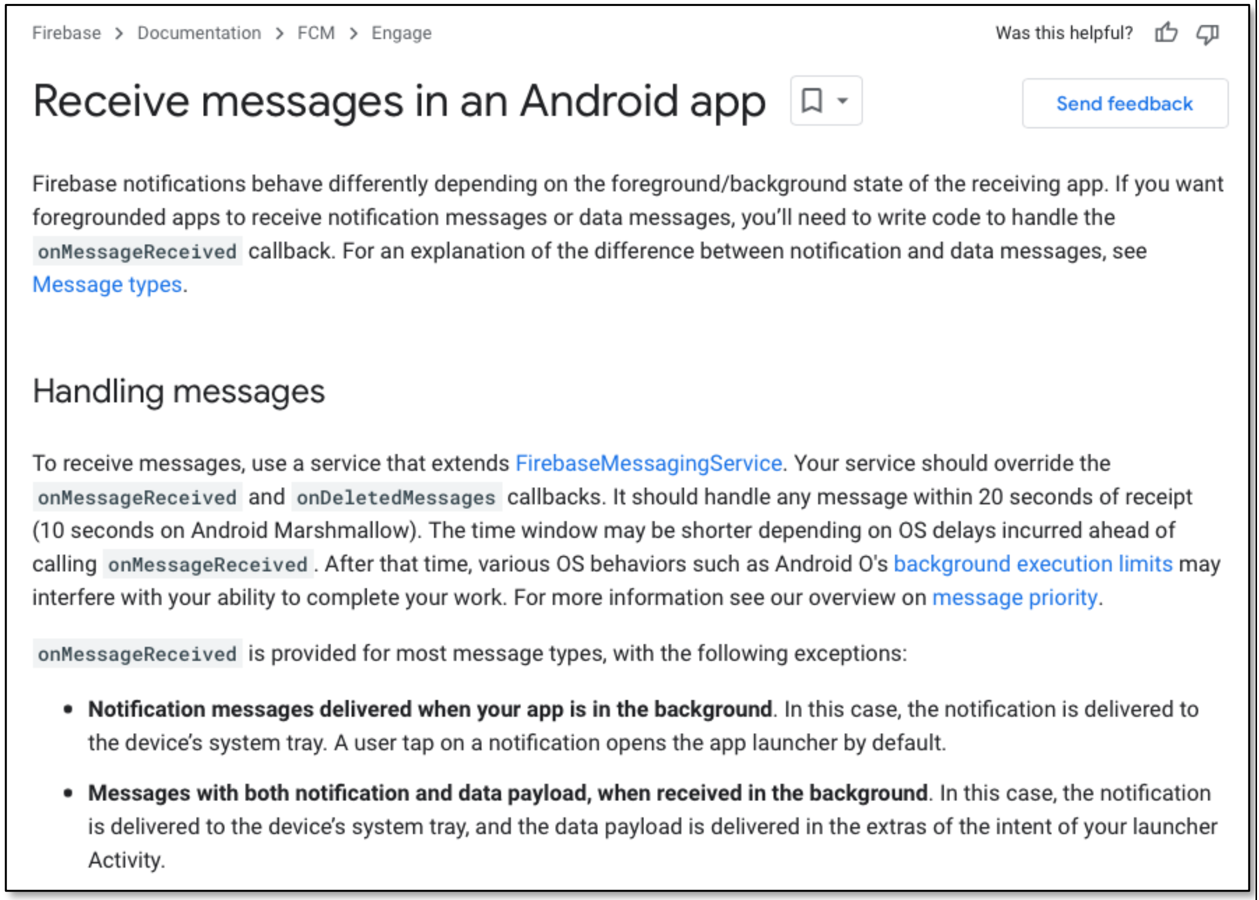


Components of Knox Manage

1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses.
2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage.
3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server.
4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on.
5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information.

https://docs.samsungknox.com/admin/knox-manage/welcome.htm

[1h] to, for each received message, map the application identifier in the message to a software process corresponding to the application identifier, and forward the application data in the message to the software process via a secure interprocess communication service.	Samsung's devices "for each received message, map the application identifier in the message to a software process corresponding to the application identifier, and forward the application data in the message to the software process via a secure interprocess communication service." For example, Samsung devices map the identifier in the message to a corresponding software process and forward the data to the application via a secure service. *See e.g.*,  Access the device registration token On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`. This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token. The registration token may change when: • The app is restored on a new device • The user uninstalls/reinstall the app • The user clears app data. https://firebase.google.com/docs/cloud-messaging/android/client;

Firebase > Documentation > FCM > Engage

Was this helpful?

Receive messages in an Android app

Send feedback

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see Message types.

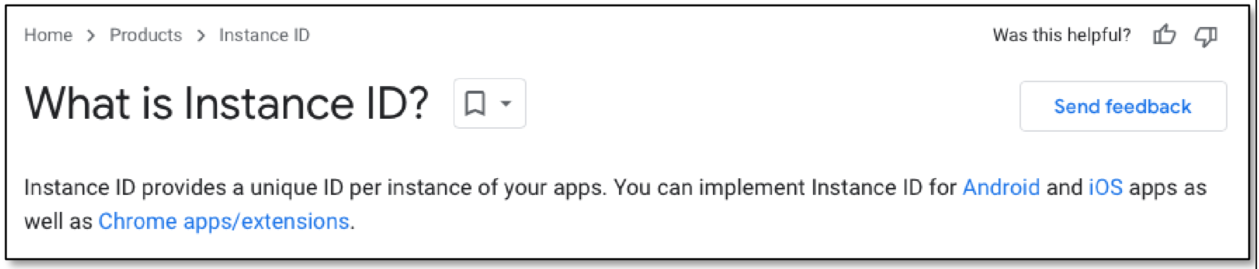
Handling messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's background execution limits may interfere with your ability to complete your work. For more information see our overview on message priority.

`onMessageReceived` is provided for most message types, with the following exceptions:

- **Notification messages delivered when your app is in the background**. In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

https://firebase.google.com/docs/cloud-messaging/android/receive;



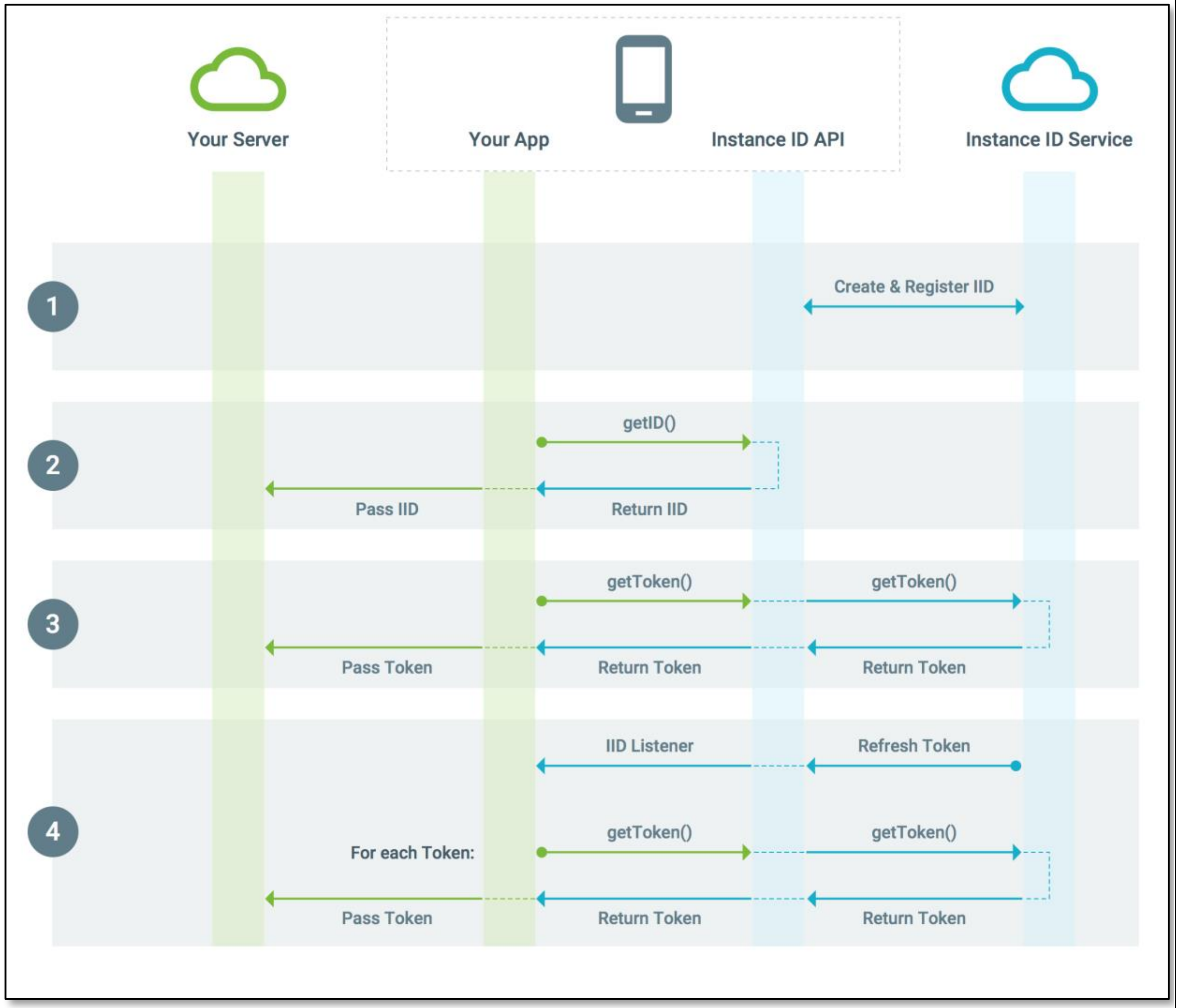
Home > Products > Instance ID

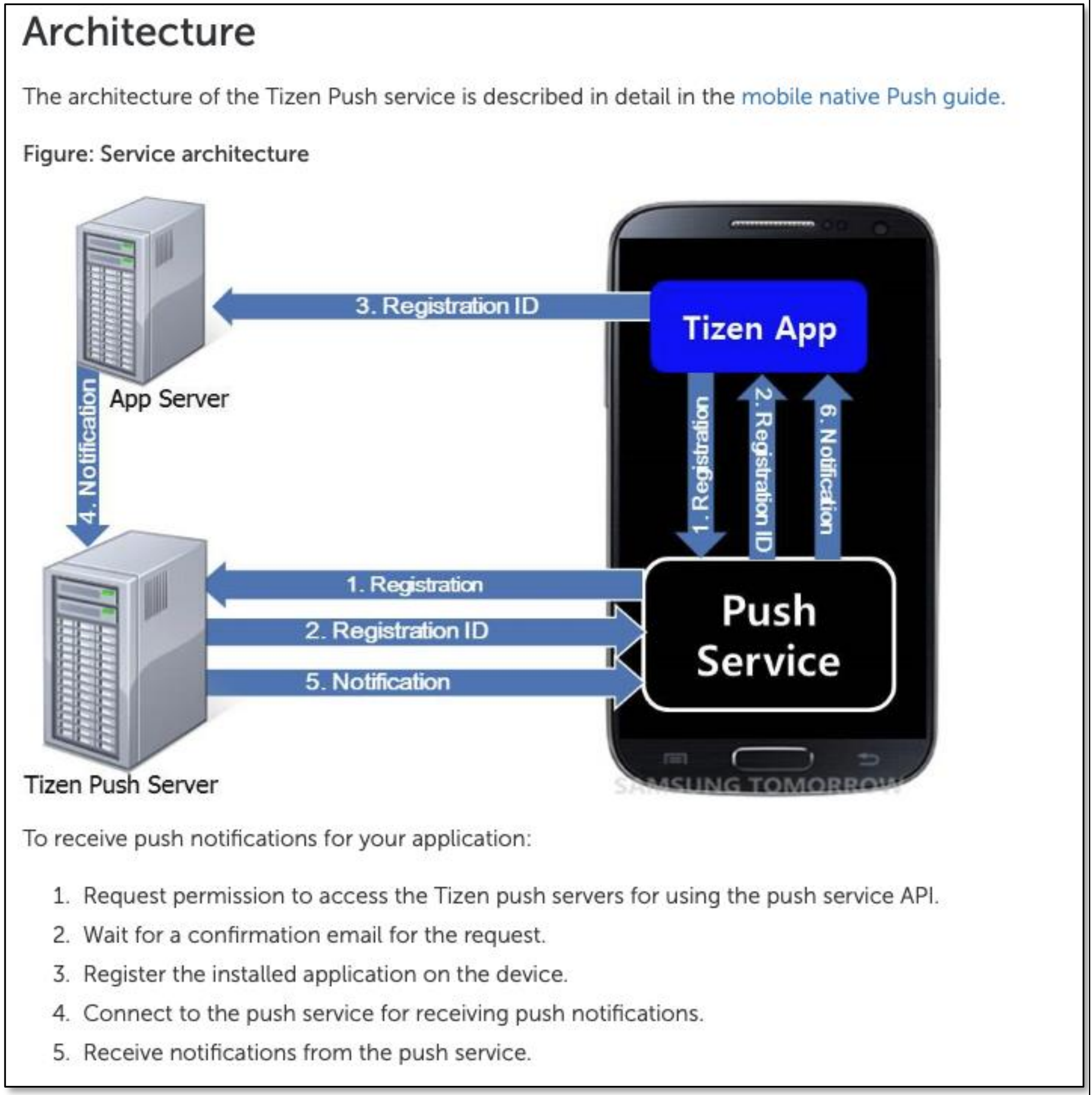
Was this helpful?

What is Instance ID?

Send feedback

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for Android and iOS apps as well as Chrome apps/extensions.

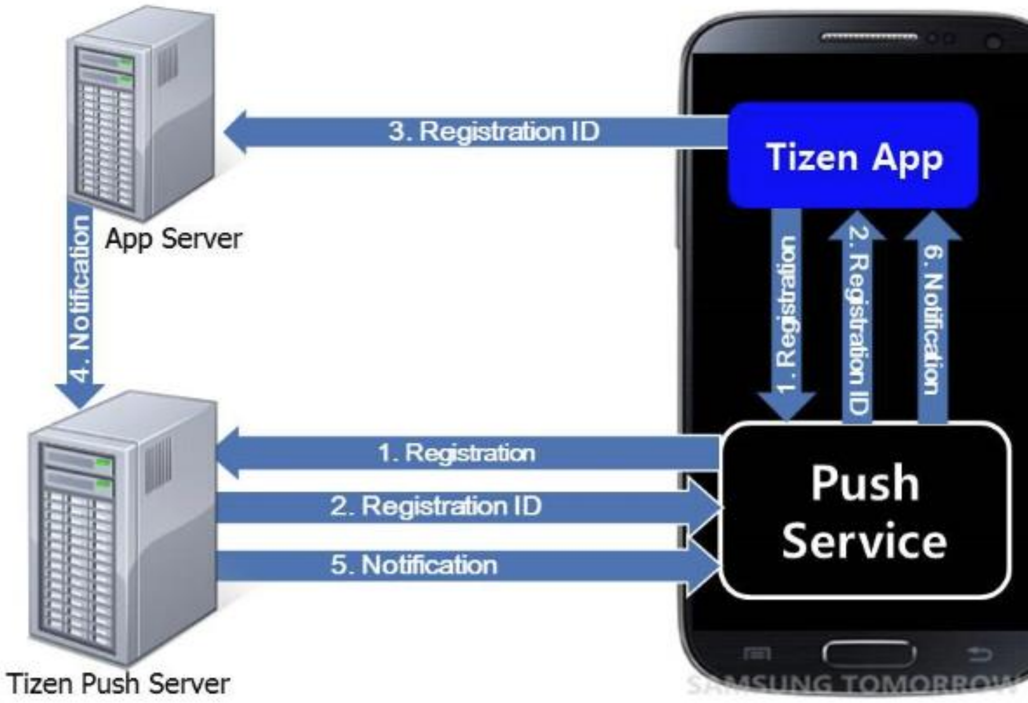
	 Your Server Your App Instance ID API Instance ID Service 1 Create & Register IID 2 getID() Pass IID Return IID 3 getToken() getToken() Pass Token Return Token Return Token 4 IID Listener Refresh Token For each Token: getToken() getToken() Pass Token Return Token Return Token https://developers.google.com/instance-id. For further example, Samsung's Tizen based devices map the identifier in the message to a corresponding software process and forward the data to the application via a secure service. *See e.g.*,

Architecture

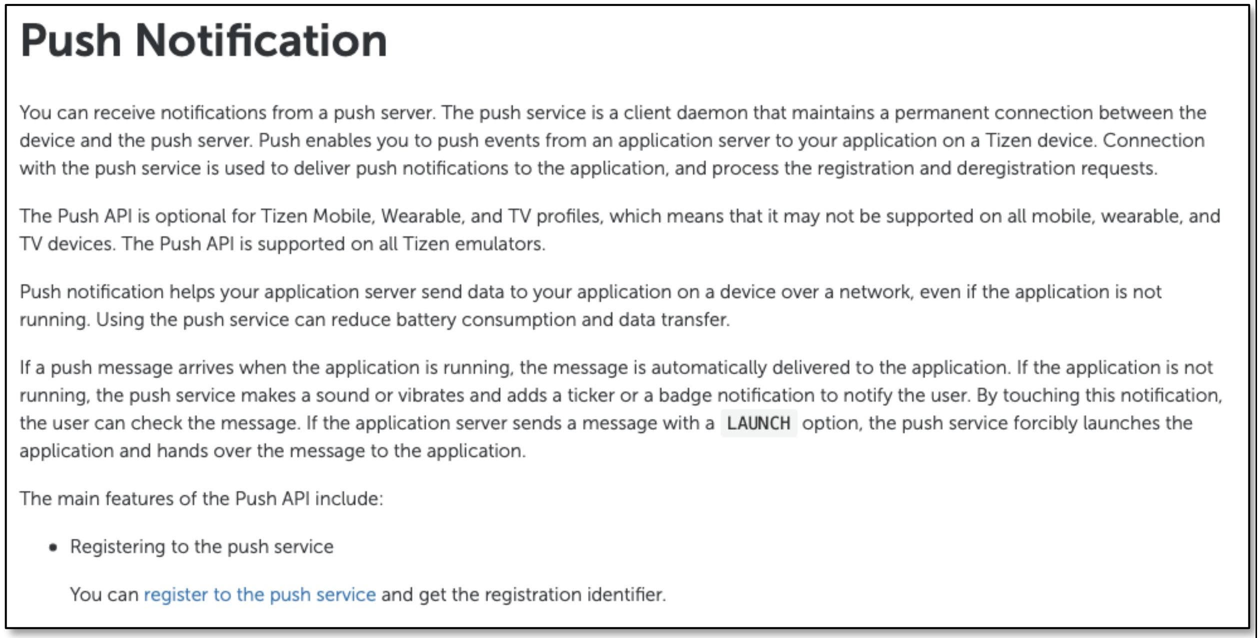
The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

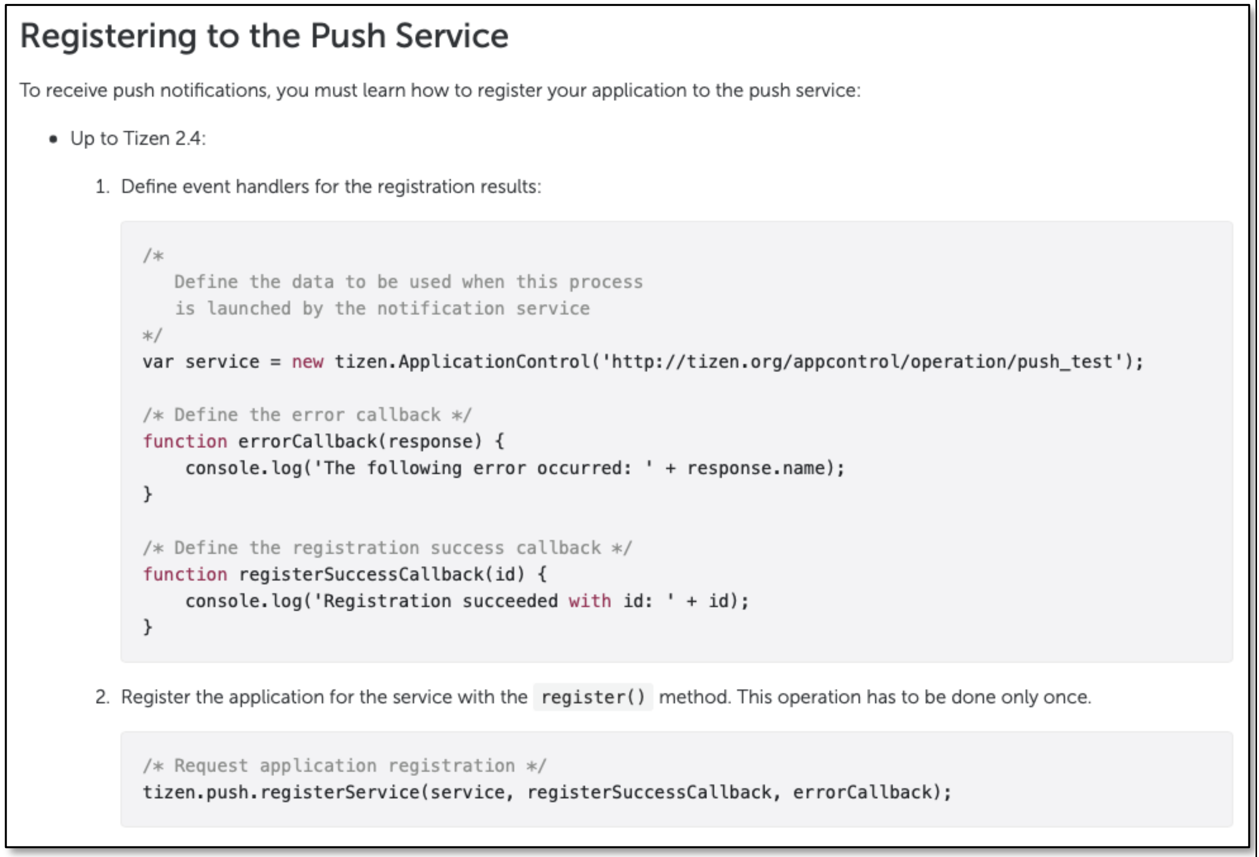
Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

  You can register to the push service and get the registration identifier.



Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

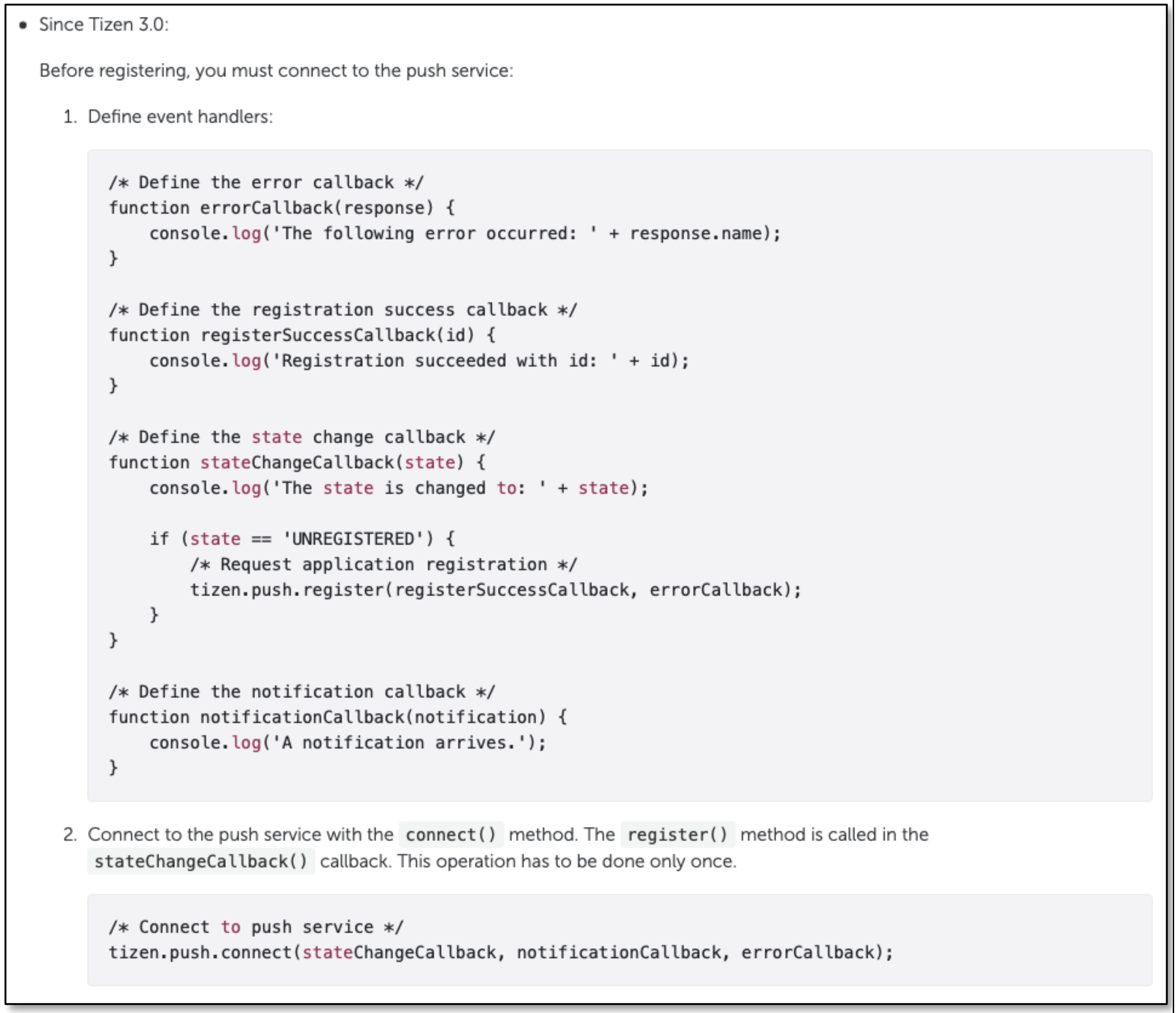
- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

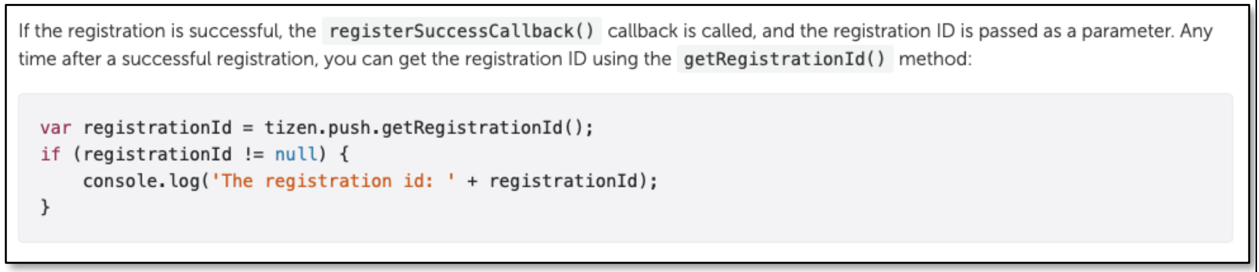
  Before registering, you must connect to the push service:

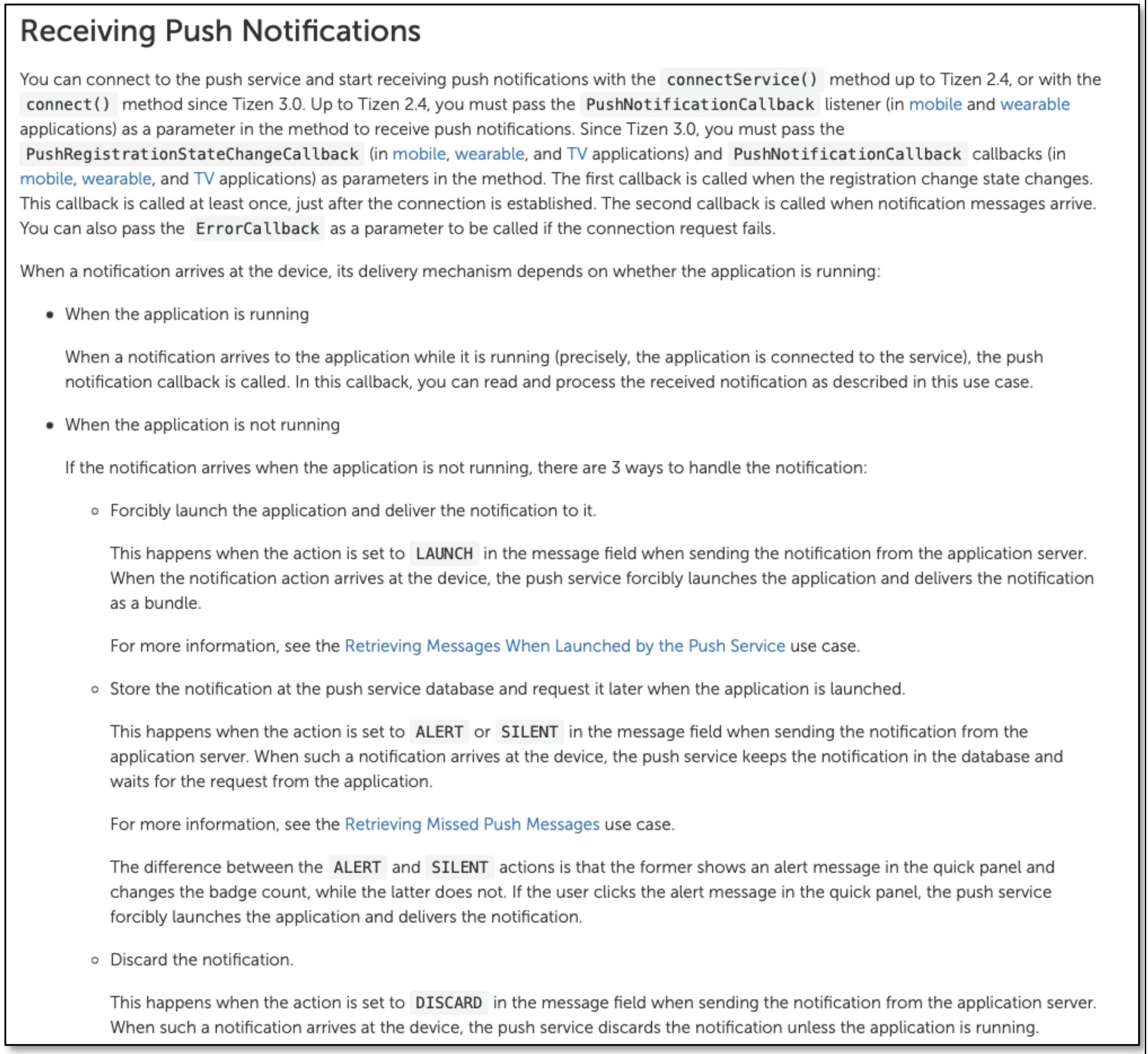
  1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

	 If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:
	`var registrationId = tizen.push.getRegistrationId();`
	`if (registrationId != null) {`
	`    console.log('The registration id: ' + registrationId);`
	`}`



Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in mobile and wearable applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the `PushRegistrationStateChangeCallback` (in mobile, wearable, and TV applications) and `PushNotificationCallback` callbacks (in mobile, wearable, and TV applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

  When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

  If the notification arrives when the application is not running, there are 3 ways to handle the notification:

  - Forcibly launch the application and deliver the notification to it.

    This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

  - Store the notification at the push service database and request it later when the application is launched.

    This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

    For more information, see the Retrieving Missed Push Messages use case.

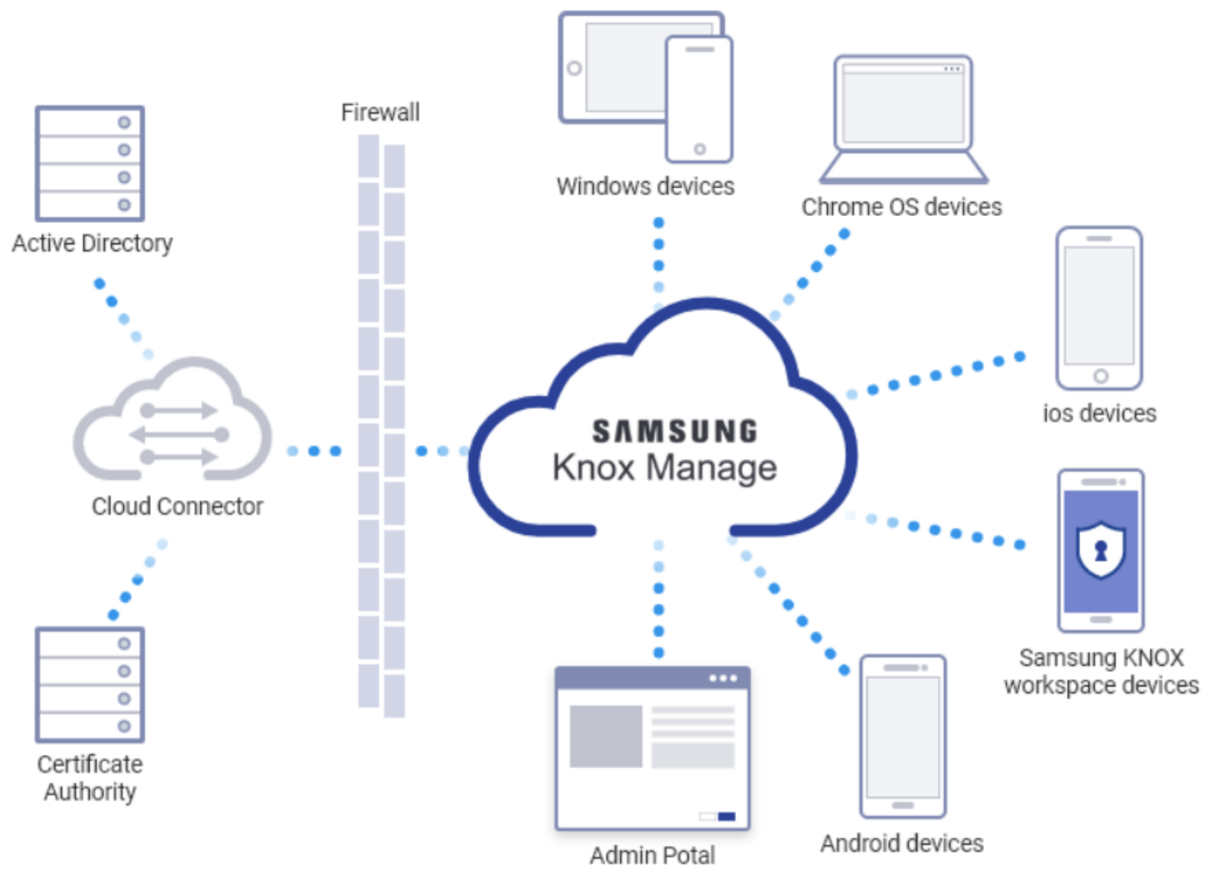
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

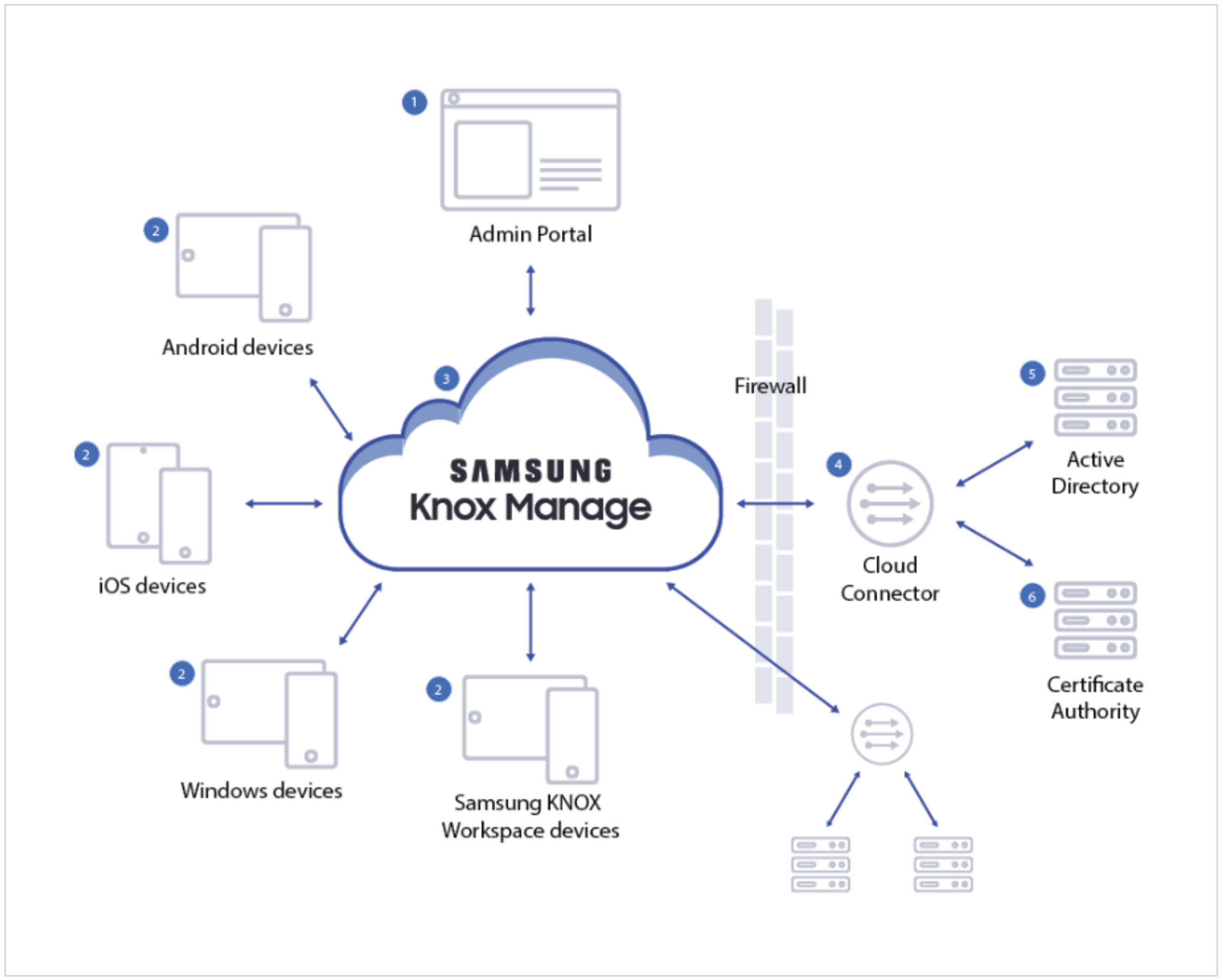
  - Discard the notification.

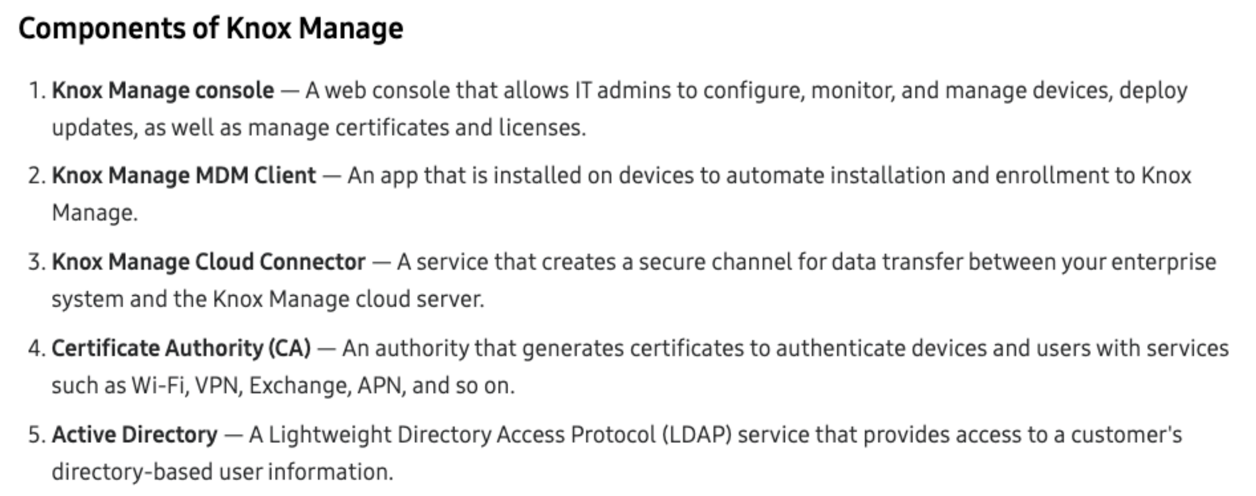
    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

https://docs.tizen.org/application/web/guides/messaging/push/.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the

	Samsung Knox servers managing those phones and devices, comprise mapping the identifier in the message to a corresponding software process and forward the data to the application via a secure service.  Active Directory Firewall Windows devices Chrome OS devices ios devices Cloud Connector SAMSUNG Knox Manage Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 1 Admin Portal 2 Android devices 2 iOS devices 3 SAMSUNG Knox Manage 2 Windows devices 2 Samsung KNOX Workspace devices Firewall 4 Cloud Connector 5 Active Directory 6 Certificate Authority https://docs.samsungknox.com/admin/knox-manage/km-features.htm

	 **Components of Knox Manage** 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm Each Tizen, Android, and Knox-managed device has an operating system (Tizen OS, Android OS, etc.) in which applications and software agents on the device are able to communicate with one another through the use of a communications bus or "a secure interprocess communication service." In the case of Android (and Knox-managed) devices, that service may be provided by the Android API; in Tizen devices, it may be provided by the Tizen API used by applications in the Tizen OS (e.g., ethe mobile and wearable Application APIs, see, e.g., https://docs.tizen.org/application/native/index). Messages received by the push service running on Tizen and Android devices, for example, communicate with other applications on the device, including sending notifications to applications that were received via the push service. Likewise, with Knox, messages sent to Knox agents from Knox servers may be authenticated by the user device and passed on to relevant application on the device or, if the application is not installed, causes an application to be installed (see, e.g, https://docs.samsungknox.com/admin/knox-manage/add-applications-intro.htm, https://docs.samsungknox.com/admin/knox-manage/assign-internal-applications.htm).